LOW-DIAMETER CLUSTERS IN NETWORK ANALYSIS

A Dissertation

by

OLEKSANDRA YEZERSKA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Sergiy Butenko
Committee Members,	Erick Moreno-Centeno
	Lewis Ntaimo
	Sivakumar Rathinam
Head of Department,	Mark Lawley

December 2016

Major Subject: Industrial Engineering

ABSTRACT

In this dissertation, we introduce several novel tools for cluster-based analysis of complex systems and design solution approaches to solve the corresponding optimization problems. Cluster-based analysis is a subfield of network analysis which utilizes a graph representation of a system to yield meaningful insight into the system structure and functions. Clusters with low diameter are commonly used to characterize cohesive groups in applications for which easy reachability between group members is of high importance. Low-diameter clusters can be mathematically formalized using a clique and an s-club (with relatively small values of s), two concepts from graph theory. A clique is a subset of vertices adjacent to each other and an s-club is a subset of vertices inducing a subgraph with a diameter of at most s. A clique is actually a special case of an s-club with $s = 1$, hence, having the shortest possible diameter.

Two topics of this dissertation focus on graphs prone to uncertainty and disruptions, and introduce several extensions of low-diameter models. First, we introduce a robust clique model in graphs where edges may fail with a certain probability and robustness is enforced using appropriate risk measures. With regard to its ability to capture underlying system uncertainties, finding the largest robust clique is a better alternative to the problem of finding the largest clique. Moreover, it is also a hard combinatorial optimization problem, requiring some effective solution techniques. To this aim, we design several heuristic approaches for detection of large robust cliques and compare their performance.

Next, we consider graphs for which uncertainty is not explicitly defined, studying

connectivity properties of 2-clubs. We notice that a 2-club can be very vulnerable to disruptions, so we enhance it by reinforcing additional requirements on connectivity and introduce a biconnected 2-club concept. Additionally, we look at the weak 2-club counterpart which we call a fragile 2-club (defined as a 2-club that is not biconnected). The size of the largest biconnected 2-club in a graph can help measure overall system reachability and connectivity, whereas the largest fragile 2-club can identify vulnerable parts of the graph. We show that the problem of finding the largest fragile 2-club is polynomially solvable whereas the problem of finding the largest biconnected 2-club is NP-hard. Furthermore, for the former, we design a polynomial time algorithm and for the latter - combinatorial branch-and-bound and branch-and-cut algorithms.

Lastly, we once again consider the s-club concept but shift our focus from finding the largest s-club in a graph to the problem of partitioning the graph into the smallest number of non-overlapping s-clubs. This problem can not only be applied to derive communities in the graph, but also to reduce the size of the graph and derive its hierarchical structure. The problem of finding the minimum s-club partitioning is a hard combinatorial optimization problem with proven complexity results and is also very hard to solve in practice. We design a branch-and-bound combinatorial optimization algorithm and test it on the problem of minimum 2-club partitioning.

DEDICATION

In loving memory of my teacher and dearest friend Vera Dmitrievna Kuzmina.

ACKNOWLEDGEMENTS

Words cannot express my gratitude to my adviser Dr. Sergiy Butenko for continuously supporting, motivating and guiding me through the struggles of graduate school and research while pursuing my PhD. He has been a great mentor and has helped me grow professionally and personally. It was a great honor to be his student and I cannot think of a better role model. I wish to also thank the members of my committee, Dr. Erick Moreno-Centeno, Dr. Lewis Ntaimo and Dr. Rathinam Sivakumar for their valuable pieces of advice and insights that have helped me improve my research projects.

I would also like to express my sincerest appreciation to Foad Mahdavi Pajouh for his fruitful collaboration and brilliant ideas. He is an extraordinary researcher and a great friend, and I am genuinely grateful for his invaluable contribution and enormous amount of help.

I wish to thank all my labmates, and the friends I made in College Station and during my summer internships at REEF for their friendship. My warmest thanks to Matthew Norton for being there during the last year of my PhD journey with endless positivity. I am also grateful to Austin Buchanan, who always patiently listened to and answered my countless questions. Additionally, I wish to thank all my friends in Ukraine, who despite distance have constantly encouraged and believed in me.

Finally, my deepest and utmost appreciation goes to my parents who made this whole thing possible. Their unconditional love, patience and support have given me strength not to give up and to accomplish my goals.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

This dissertation is comprised of topics dedicated to various techniques for analyzing complex systems using graph-theoretic tools. Almost any system can be conveniently modeled as a graph, or network, $G = (V, E)$, where V is a set of vertices (nodes) used to represent system entities, and E is a set of edges (arcs) used to describe certain relations between these entities.

1.1 Cluster-based network analysis

Network analysis is an interdisciplinary field that utilizes graph-theoretic techniques to perform a quantitative and structural analysis of the systems modeled as graphs. Cluster-based analysis is one of the most common and prominent fields of network analysis. It can be classified into two classes: detection of the largest clusters in the graph and partitioning the graph into clusters. These techniques allow to reveal hidden patterns and understand the functions of the underlying systems. Moreover, the size of the largest cluster in the graph can indicate an overall system cohesiveness, while partitioning the graph into tight clusters in some applications can help reduce the size of the graph and obtain an impression of its hierarchical structure.

There are many ways to define a cluster. One of the earliest mathematical formalizations of a cluster is a *clique* [51], introduced in 1949 by social scientists to represent a cohesive, or "tightly knit", subgroup in a social network, e.g., a group of people who all know each other. In graph-theoretic terms, a clique is a subset of pairwise adjacent vertices. It is a "perfect" cluster in that it possesses all desirable properties of a tight cluster, such as highest familiarity between the members of a cluster (i.e., highest vertex degree), highest connectivity and robustness to failures

(largest number of edges), and highest reachability and access between the members of a cluster (shortest pairwise distances and diameter). Ever since the formal concept of a clique was introduced, it has become the classical model for a cluster and has garnered much attention among researchers from many fields apart from just the social sciences. Finding large cliques in a graph that represents a particular system can help shed a light on the underlying structure of the communities which comprise the system. Thus, the clique concept, although simple, led to insightful application of large clique detection in a wide variety of fields, with some of the earliest including applications in computer vision, experimental design, information retrieval, coding theory, fault diagnosis, social network analysis, telecommunication and computational biochemistry and genomics among other areas [60, 75, 14, 7].

Prior applications of clique analysis and related studies rely heavily on the assumption that all nodes and edges are known or assumed to exist with certainty (i.e analysis of deterministic graph structures). However, a simple deterministic graph might not always properly reflect the complexities and uncertainties of the underlying system. Consider, for example, a graph representing Facebook friendships. Large cliques in Facebook graph are likely to represent communities of people associated via common affiliations, activities, or interests, with many weak connections included in this structure. However, one may be more interested in finding groups of people forming strong friendship circles (i.e., groups characterized by strong connections). One possible way of approaching this problem is to associate each edge with the probability that it represents a "true" friendship in real life, so the complement of such probability can be thought of as the likelihood of failure of the corresponding edge (i.e., the probability that an edge is a strong connection). Then one can look for large *robust* cliques (introduced in Chapter 2), whose cohesiveness properties are not affected significantly (as quantified by certain risk requirements) by the possible edge

failures. One would expect that the stricter the risk requirements are, the smaller in size the solution is. Similarly, in real life, the less we are concerned with the strength and quality of the friendship, the larger groups of friends we can observe. On the other hand, the more reliable and close a group of friends is, the smaller it is in size. Examples of uncertain graphs that could benefit from robust clique detection analysis are not limited to social networks. Other examples where the concept of a robust clique can be meaningfully applied include communication and transportation networks where edges may fail due to random disruptions or targeted attacks, some biological networks where edges may exist with a certain probability associated with the errors of the data extraction process, financial networks where edges correspond to correlation between the stocks and may exist with a certain confidence level based on the accuracy of historical data and forecasts.

Despite the popularity of the clique concept and its wide range of applications, it has a, potentially, major drawback. Specifically, clique structure can often be overly restrictive, with a formal definition that is quite inflexible, reflecting a type of idealistic structure. Thus, a clique (or the associated analysis based upon cliques) may prove incapable of representing the imperfections of real-life networks. This potential drawback has motivated the development of various alternative, less restrictive, representations of a cluster. These are referred to as clique relaxations, as they relax one or several defining clique properties, such as degree, distance (diameter), and edge density. Different types of these relaxation models have been proposed in the literature and have found many applications in various domains [62].

A distance-based relaxation of a clique, called an s-club, restricts the size of the diameter of the resulting subgraph and is the topic of Chapters 3 and 4 in this dissertation. Low-diameter structure is very common and important in transportation and telecommunication networks, where fast and easy access to all entities is of high

importance. Clique, or a 1-club, is the structure with the lowest possible diameter, however designing it might be too expensive. An s-club with a small value of s is a good alternative to clique, as it ensures relatively small diameter (no more than s) and easy reachability between the nodes. However, since this structure disregards other important clique properties, such as robustness and connectivity, it might be vulnerable to node and edge failures in some applications. For example, an extreme case of a 2-club – a *star*, which is a graph with one "hub" vertex connected to the rest of the vertices that are not directly connected to each other, is apt to complete disconnection due to the failure of the hub [62]. Therefore, in the situations where failures are anticipated, it is important to design or detect clusters not only of small diameter, but also with additional constraints on their connectivity. In Chapter 3, we study connectivity properties of 2-clubs and introduce a *biconnected 2-club* model that ensures the existence of at least two vertex-disjoint paths between any two nodes in this 2-club. That way, if an edge or a node on one of the paths fails, the rest of the nodes are still connected to each other. Additionally, we study a *2-club* that is not biconnected which we call *fragile*. The size of the largest fragile 2-club in the network may provide insight into network vulnerabilities and its susceptibility to disconnection under failures and disruptions.

Another very powerful and common tool in analyzing complex networks is clustering. The goal of clustering is to arrange the data elements into tight groups where members of each group are "highly similar." Such an approach helps scale down the data and obtain its hierarchical structure as well as reveal hidden patterns and derive important properties of the underlying systems. Graph-based clustering has been studied extensively and an abundance of various techniques and methods are available in the literature [29].

Most existing clustering techniques are based on partitioning a graph into clusters

in such a way that a certain measure of density and connectivity within the clusters and sparsity between them is optimized. One such measure, which is among the most famous, is the *modularity* introduced by Newman and Girvan in 2004 [57]. Modularity represents the difference between the coverage, i.e., the fraction of the edges that fall within the clusters, and the expected coverage if the edges were placed randomly. A larger value is believed to correspond to better clustering. Finding a clustering with maximum modularity is an NP-hard problem, therefore, for large-scale instances of graphs, it can only be approached with heuristic algorithms, with many already proposed in the literature [29]. There exist many other measures that have been used to find a graph clustering and estimate its quality. Like modularity, most of these correspond to NP-hard problems. Although these approaches have many applications, none of these techniques are able to guarantee that all detected clusters have the same topological structure.

Clique-based clustering, on the other hand, is an approach that partitions a graph in such a way that each component is a clique. The clique partitioning problem has been used to model a variety of problems arising in computational biology [45], transportation [24], telecommunication [19, 47], etc. Partitioning into cliques, however, is not always appropriate due to the restrictive nature of a clique, as it was already mentioned earlier. Clique structure fails to account data imperfection that might result in missing information about vertices and edges. Moreover, many naturally arising cohesive subgroups are not perfectly interconnected and may be a few connections short of a clique structure. Consider, for example, a network of protein interactions where vertices represent proteins and edges between two vertices exist when the corresponding proteins are known to interact with each other. Usually, tight clusters in such networks correspond to certain functional complexes. Analysis of protein interaction networks and finding such clusters help scientists to derive

hypotheses regarding the function of certain proteins or groups of proteins. Fig. 1.1 shows a few examples of known protein functional complexes [49, 62]. Note that none of these clusters are cliques, either due to the underlying nature of given protein interactions, or due to the absence of evidence that a certain pair of proteins interact with each other. In fact, all the subgraphs induced by the vertices belonging to a certain protein complex have a diameter equal to 2 or 3, so they can be modeled as diameter-based relaxations of a clique. Particularly, the subgraph induced by the set of vertices in the middle, corresponding to the 1dxr (photosynthesis) protein complex, has a diameter of 3 and a structure of a 3-club. Similarly, the rest of the subgraphs in Fig. 1.1 have a diameter of 2 and are 2-clubs. Therefore, in the analysis of protein interaction networks, an s-club partitioning technique may be a suitable alternative to a clique partitioning. Moreover, an s-club partitioning can also be beneficial in applications where short cluster diameter is a critical characteristic, such as in wireless sensor networks [2, 47, 28].

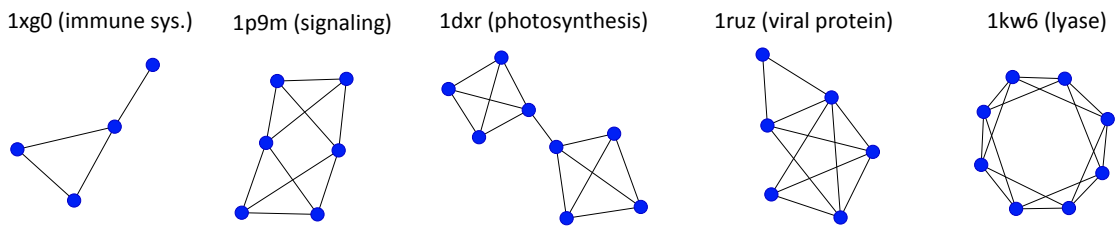


Figure 1.1: Examples of protein functional complexes [49, 62].

An s-club and many other clique relaxations have been studied from different prospectives, theoretical and optimization-based [62]. Most of these studies, though, are focused on the problem of detecting a certain clique relaxation of the maximum size. Little to no research has been done in the area of clustering the graph into

clique relaxations. We attempt to start filling this gap in Chapter 4 by studying the problem of minimum s-club partitioning and designing the algorithms to solve it.

1.2 Graph theory notations and definitions

Throughout this dissertation, we consider a finite simple undirected graph $G = (V, E)$, where $V = \{1, \ldots n\}$ and $(i, j) \in E$ if vertices $i, j \in V$ are adjacent. The *open neighborhood* of a vertex i, denoted by $N_G(i)$, is the set of all vertices adjacent to i in G. The set $N_G(i) \cup \{i\}$ is referred to as the *closed neighborhood* of i in G and is denoted by $N_G[i]$. The *degree* of vertex i in G, denoted by $deg_G(i)$, is the size of its open neighborhood, $|N_G(i)|$. The *distance* between vertices u and v in graph G is defined as the length of a shortest path between them in G and is denoted as $d_G(u, v)$. The *diameter* of G is then the largest distance in G, $diam(G) = \max_{u,v \in V} d_G(u, v)$. The *(vertex) connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices that need to be removed from the graph to disconnect it. The (vertex) connectivity also corresponds to the smallest number of vertex-disjoint paths between each pair of vertices. A graph G is said to be *k-connected* if $\kappa(G) \geq k$. Also, for any set $S \subseteq V$, the subgraph induced by S is defined as $G[S] = (S, E \cap (S \times S))$.

Given $G = (V, E)$, a *clique* is defined as a subset $C \subset V$ of mutually adjacent vertices. A graph is called *complete* if the set of its vertices forms a clique. A *maximal clique* is a clique that can not be extended to a clique of larger size. The *maximum clique problem* asks for a clique of the largest cardinality in the graph. The size of a maximum clique is called the *clique number* of G and is usually denoted as $\omega(G)$.

An *independent set* (or *stable set*) is a subset $I \in V$ of vertices with no edges between them. It is easy to see that a *clique* in G is an *independent set* in the *complement graph* of G, $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(i, j) : (i, j) \notin E \; \forall i, j \in V, i \neq j\}$, and vice versa. Finding a maximum clique in G is equivalent to finding a maximum

independent set in $\bar{G}$. The cardinality of a maximum independent set in G is called the *independence* or *stability number* of G, denoted $\alpha(G)$, and $\alpha(\bar{G}) = \omega(G)$.

A *proper vertex coloring* (also referred to as a coloring for simplicity) is an assignment of a color (or a label) to each vertex in a graph such that no adjacent vertices have the same color. More formally, given a set of colors $\{1, 2, ..., k\}$, a *proper k-coloring* is a mapping $f : V \to \{1, 2, ..., k\}$, where $f(v_i) \neq f(v_j)\ \forall (i, j) \in E$. Then the vertices with the same value of f belong to the same *color class*. The *chromatic number* of a graph G, denoted $\chi(G)$, is a minimal number of colors necessary to color G. An *optimal coloring* of a graph is a proper coloring that uses exactly $\chi(G)$ colors.

It is easy to see that the vertices that belong to the same color class form an independent set, and the vertices of a clique all belong to different color classes. Then it follows that for a given graph G:

$$\chi(G) \geq |V|/\alpha(G),$$

and

$$\chi(G) \geq \omega(G).$$

The maximum clique and vertex coloring problems often arise in similar applications, where they play complementary roles. For example, let the vertices in the graph represent elements of a system and the edges between them – the incompatibility between those elements. Then a maximum clique will represent a set of mutually incompatible elements of the largest size in the system, whereas, the color classes will correspond to sets of mutually compatible elements. The coloring problem is also equivalent in the complement graph to the problem of *minimum clique partitioning*, which is to partition a graph into the smallest number of cliques.

The maximum clique and vertex coloring problems are well-known NP-hard problems [31, 44] that are hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ [4, 5, 39, 25, 79].

Other cluster formalizations, known as *clique relaxations* have been proposed in the literature to mitigate restrictive nature of a clique concept. The focus of this dissertation is the distance-based clique relaxation, called an s-club. A subset of vertices $S \subseteq V$ is called an s-club [56] if it induces a subgraph $G[S]$ with $diam(G[S]) \leq s$. The maximum s-club problem, which is to find an s-club of the largest cardinality in the graph, is NP-hard for any fixed s [12], even when restricted to graphs of diameter $s + 1$ [8]. Moreover, it is also NP-hard to test s-club maximality for any fixed integer $s \geq 2$ [52]. The minimum s-club partitioning problem which aims to partition the graph into the smallest number of non-overlapping s-clubs is NP-hard [20], even when restricted to bipartite and chordal graphs [1, 16].

Other examples of clique relaxations mentioned in this dissertation are:

- an *s-clique*, which is a structure similar to an s-club, but less restrictive, and is defined as a subset of vertices $S \subseteq V$ where $d_G(u, v) \leq s$ for all $u, v \in S$.
- an *s-defective clique* [77], which is a subset of vertices $S \subseteq V$ inducing a complete subgraph missing s edges.
- an *s-plex* [66], which is a subset $S \subseteq V$ of vertices inducing a subgraph with the minimum degree at least $|S| - s$.

1.3 Structure of dissertation

In this dissertation, we contribute to the field of network analysis by introducing several novel tools for cluster-based network analysis. To address uncertainty and vulnerability of some low-diameter cluster models, we formalize and study their

robust counterparts and additionally design the solutions techniques to solve their corresponding optimization problems. More specifically, we introduce a robust clique model and study the problem of largest robust clique detection in Chapter 2. Then, in Chapter 3, we study connectivity properties of 2-clubs and introduce a notion of biconnected and fragile 2-clubs. Solution approaches to find the largest biconnected and fragile 2-clubs are also presented in Chapter 3. Additionally, we study the problem of the minimum s-club partitioning in Chapter 4. For all these mentioned problems, we perform computational experiments to test performance of the proposed solution techniques and report obtained results. Finally, we conclude this dissertation in Chapter 5 by summarizing our contributions and outlining the future directions of research.

2. DETECTING ROBUST CLIQUES IN GRAPHS SUBJECT TO UNCERTAIN EDGE FAILURES[1]

In this chapter, we will introduce a *robust* clique, a variation of clique model for graphs subject to multiple uncertain edge failures. The desired robustness properties are enforced using Conditional Value-at-Risk (CVaR) risk measure. We will further design solution techniques for computing approximate solutions to the corresponding problem of finding the maximum robust clique. More specifically, we develop and compare several heuristic approaches, as well as an exact combinatorial branch-and-bound algorithm. The proposed heuristics are adaptations of the well-known tabu search and GRASP methods, whereas the exact approach is an extension of Östergård's algorithm [58] for the maximum clique problem. We will also perform the computational experiments comparing all approaches on DIMACS graph instances and report the results in the end of this chapter.

2.1 Introduction

As it was already mentioned in Chapter 1, a clique is an earliest and most common model used to describe cohesive subgroups. The clique itself and the corresponding optimization problem of finding a clique of the maximum size in a given graphs have been extensively studied in the literature [10, 75]. However, most of the previous work have been mainly focused on the deterministic case, where all vertices and edges of the input graph G are assumed to be known and certain. Yet, this assumption may not be realistic for many applications, where some vertices and/or edges can fail or are not known to exist with certainty. This motivates one to examine cliques

[1]Reprinted with permission from "Detecting robust cliques in graphs subject to uncertain edge failures" by O. Yezerska, S. Butenko, V. Boginski, 2016. *Annals of Operations Research*, doi: 10.1007/s10479-016-2161-0, Copyright [2016] by Springer.

not only in terms of their size, but also with respect to their robustness to potential failures.

The problems dealing with detection of large clusters in networks subject to uncertain failures are only beginning to be addressed in the literature. However, in the recent years there has been a substantial amount of studies related to uncertain graph analysis from other viewpoints. In particular, Hintsanen et al. [40] introduced the problem of finding the most reliable subgraph in the uncertain (probabilistic) graph subject to edge failures, which is a subgraph with the largest probability of remaining connected after the failures have occurred. Similar problems have been consequently studied in [41, 42]. The problem of graph-based clustering (partitioning), where the input graph is uncertain, was considered in [46, 50]. Namely, in [50] Liu et al. studied the problem of finding reliable clusters with the largest probabilities of not being disconnected, whereas in [46] Kollios et al. detected clusters using an expected edit distance metric.

In a study devoted to finding top-k maximal cliques in graphs subject to uncertain vertex and edge failures, Zou et al. [78] developed a branch-and-bound algorithm that detects k sets of vertices with the highest *maximal-clique* probability, which is the probability that such sets will form maximal cliques. In two papers most closely related to our work, the versions of the maximum clique problem were studied in graphs subject to different types of uncertainty. Particularly, Rysz et al. [65] considered graphs subject to random vertex failures and developed a branch-and-bound algorithm for finding the risk-averse maximum weighted clique, where risk was measured using a family of coherent risk measures that included CVaR. In the other paper, Miao et al. [54] considered the maximum probabilistic clique problem in graphs with random edge failures, which asks for a subset of vertices of the largest cardinality forming a clique with a specified probability.

In this chapter we consider the problem of finding the largest clique satisfying given robustness requirements (expressed in terms of CVaR) in graphs with uncertain edge failures. The novelty and the most important part of the proposed approach lies within its ability to control certain desirable properties of the structure resulting from a clique after failures have occurred. In addition, to the best of our knowledge, this is the first work proposing metaheuristic approaches to clique detection in networks involving uncertainties. Using metaheuristics is motivated by the fact that the considered problem is NP-hard (which follows from the observation that the structures sought posses the property of heredity in the induced subgraphs [76, 54]). Moreover, our experiments with an extended version of Östergård's algorithm for the maximum clique problem suggest that even small instances of the problem are extremely difficult to solve.

Hence, to tackle the problem of interest, we present three metaheuristics, which are adaptations of the well-known tabu search (TS) [34, 35, 36], STABULUS (TS's variation for finding stable sets) [30], and GRASP [26, 27] algorithms. The main idea behind TS is to prevent getting trapped in the local optima by allowing non-improving moves, and prohibit obtaining repetitive solutions using the so-called *tabu lists* and *tabu tenures*. Variations of TS have been successfully applied to the maximum clique problem [32, 69, 73, 74] and have shown excellent performance. The main difference between TS and STABULUS is that in the latter one the local search is performed on partially infeasible solutions. Once the feasible solution is found, the algorithm restarts in an attempt to find a solution of better quality. GRASP is a multi-start algorithm, where in each restart, it constructs an initial solution in a greedy randomized fashion and then attempts to improve it using local search techniques. To construct an initial solution, the elements to be included are picked randomly from the so-called *restricted candidate list* (RCL), which contains some of

the best candidates. RCL construction is the greedy part of the algorithm whereas the selection rule is the random one.

The chapter is organized as follows. In the next section we describe how the uncertainty associated with the edge failures can affect clique properties and how it can be modeled and managed when finding a robust clique. Section 2.3 covers some of the key techniques used in all three heuristics. The heuristics themselves are discussed in detail in Sections 2.4-2.6. We test and compare heuristics' performance and present the results in Section 2.7.

2.2 Risk quantification in cliques

Consider a simple graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, subjected to uncertain edge failures, where each edge $(i, j) \in E$ fails with probability p_{ij}, independently of other edges. Formally, given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the set of possible outcomes, $\mathcal{F}$ is the σ-algebra representing the set of events, and $\mathbb{P}$ is the probability measure $\mathbb{P} : \mathcal{F} \mapsto [0, 1]$, let $Y : \Omega \mapsto \{0, 1\}^{|E|}$ be a random variable associated with the edge failures, defined as follows:

$$Y_{ij} = \begin{cases} 1, & \text{with probability } 1 - p_{ij}, \\ 0, & \text{with probability } p_{ij}. \end{cases}$$

Let $G^s = (V, E^s)$ denote the graph resulting from such edge failures under realization scenario $s \in \Omega$. Then a clique C in G will correspond to some clique relaxation structure in G^s [62]. Examples of such structures include k-defective clique, which is a subset of vertices inducing a complete subgraph missing k edges [77] and k-plex, a subset C of vertices inducing a subgraph with the minimum degree at least $|C| - k$ [66].

Violation of clique properties inflicted by edge failures can be thought of as a loss

that needs to be controlled in order to ensure robustness of the remaining structure. To quantify the extent of such a loss, we can define a *loss function* $L_C(Y)$ associated with the clique C and the random variable Y. The loss function is a random variable that characterizes some structural properties of the subgraph induced by C in the post-failure graph, e.g., the number of missing edges or the maximum number of non-neighbors of a vertex. The choice of a loss function will be dictated by the structural properties expected of the subgraph induced by C. In this work, we consider the loss function that expresses the maximum number of neighbors that a vertex in C looses as a result of edge failures, that is, given a clique C, we have:

$$L_C(Y) = \max_{i \in C} \sum_{j \in C \setminus \{i\}} (1 - Y_{ij}). \tag{2.1}$$

Let $\mathcal{G}$ be the set of all real valued loss functions. Then a risk measure ρ is defined as a mapping $\rho : \mathcal{G} \mapsto \mathbb{R}$. Depending on the risk preferences and other criteria (e.g., mathematical properties or tractability), different risk measures can be used, such as expected value, excess probability, expected excess, etc. In this work, we use the Conditional Value-at-Risk (CVaR). As an asymmetric measure of risk, asymmetric in that it only considers the largest losses, and a conservative summation of these large losses, we find CVaR to be an attractive characterization of risk with intuitive interpretation. In addition, we note that other measures of risk, such as Value-at-Risk (VaR), are similar asymmetric measures of risk that may also be used. CVaR, though, is attractive due to its conservatism and in that it considers the magnitude of the large losses. VaR, for example, does not and can be far less conservative, particularly when large losses far exceed moderate losses. Thus, we find CVaR a more appropriate measure of the risk, particularly in the context of robustness. These and other desirable mathematical properties have made CVaR a

popular choice in financial engineering [59], military [48], and, more recently, network analysis [65] applications.

Intuitively, CVaR with the confidence level $\alpha \in (0, 1)$ is the expected loss over the $100(1 - \alpha)$ percent of worst case losses. In other words, it is the expectation of the losses exceeding VaR_α, where VaR_α is the α-quantile of the distribution of losses, defined as $VaR_\alpha[L] = \min\{\zeta : \mathbb{P}(L \leq \zeta) \geq \alpha\}$.

The following minimization problem can be used to compute CVaR [63, 64]:

$$CVaR_\alpha[L] = \min_{\zeta \in \mathbb{R}} \{\zeta + \frac{1}{1-\alpha}\mathbb{E}[(L - \zeta)_+]\}, \tag{2.2}$$

where $(\cdot)_+ = \max\{\cdot, 0\}$. Computing CVaR directly from (2.2) can be a challenging task. In particular, in the problem of our interest, determining the distribution of $L_C(Y)$ requires an explicit enumeration of all the possible edge failures in clique C. The complexity of such a procedure is $O(2^{m_C})$, where $m_C = |C|(|C|-1)/2$, therefore it can only be performed on small clique instances. Alternatively, if we generate a sample S of scenarios, then,

$$CVaR_\alpha[L_C(Y)] \approx \min_{\zeta \in \mathbb{R}} \{\zeta + \frac{1}{1-\alpha} \sum_{s \in S} \pi_s (L_C(Y^s) - \zeta)_+\}, \tag{2.3}$$

where Y^s is the realization of Y under the scenario s and π_s is the probability of scenario $s \in S$.

Moreover, if we assume for simplicity that all scenarios are equiprobable, that is $\pi_q = \frac{1}{|S|}$, then (2.3) can be reformulated as

$$CVaR_\alpha[L_C(Y)] \approx \min_{\zeta \in \mathbb{R}} \{\zeta + \frac{1}{|S|(1-\alpha)} \sum_{s \in S} (L_C(Y^s) - \zeta)_+\}. \tag{2.4}$$

After linearization of (2.4) we obtain:

$$\begin{aligned} CVaR_\alpha[L_C(Y)] \approx \quad \min \quad & \{\zeta + \tfrac{1}{|S|(1-\alpha)} \sum_{s\in S} z_s\}, \\ \text{s.t.} \quad & z_s \geq L_C(Y^s) - \zeta, \ \ \forall s \in S, \\ & z_s \geq 0, \ \ \forall s \in S, \\ & \zeta \in \mathbb{R}, \end{aligned} \tag{2.5}$$

which can be further reformulated as:

$$\begin{aligned} CVaR_\alpha[L_C(Y)] \approx \quad \min \quad & \{\zeta + \tfrac{1}{|S|(1-\alpha)} \sum_{s\in S} z_s\}, \\ \text{s.t.} \quad & z_s + \zeta \geq |C| - 1 - \sum_{j\in C\setminus\{i\}} Y_{ij}^s, \ \forall s \in S, \forall i \in C, \\ & z_s \geq 0, \ \ \forall s \in S, \\ & \zeta \in \mathbb{R}. \end{aligned} \tag{2.6}$$

2.3 Methodology

All of the proposed heuristics for detecting a robust clique share several common features, which we will discuss in this section before proceeding to the formal description of each heuristic. Below in this chapter, by L_C we will always mean the loss function defined in (2.1).

2.3.1 CVaR verification procedure

In this work, the solution's *robustness* is determined by verifying whether the corresponding clique satisfies the CVaR requirement:

Definition 1. *A clique C is K-$CVaR_\alpha$-robust if $CVaR_\alpha[L_C] \leq K$.*

Remark 1. *For simplicity purposes, throughout this chapter we will call a clique satisfying Definition 1 for the considered loss function L_C a robust clique.*

As mentioned in Section 2.2, computing the exact value of $CVaR_\alpha[L_C]$ is computationally challenging. Therefore, to enhance the CVaR verification procedure and avoid unnecessary cumbersome calculations, we first estimate the bounds on CVaR value.

Perhaps the simplest bounds are based on the smallest and largest edge failure probabilities, p_{min} and p_{max}, of the current solution. These bounds are extracted from the tables with precomputed values of CVaR for clique instances with the same edge failure probability (Appendix A). However, these bounds can be rather loose and may not be sufficient to decide whether the solution is robust.

A tighter lower bound on CVaR value can be calculated using the following proposition.

Proposition 1. *Given a clique* $C = \{1, 2, \ldots, |C|\}, |C| \geq 3$ *and* $v \in C$, *let* L_v *be the random variable describing the number of failed edges in the set* $\{(v, i) : i \in C\}$. *Then,*

$$CVaR_\alpha[L_C] > \max_{v \in C} CVaR_\alpha[L_v]. \tag{2.7}$$

Proof. The event $\{L_C = |C| - 1\}$ occurs when at least one of the vertices loses all its incident edges in the subgraph $G[C]$ induced by C. Then,

$$\begin{aligned}
&\mathbb{P}\{L_C = |C| - 1\} \\
&= \mathbb{P}\{L_v = |C| - 1\} + \mathbb{P}\{L_C = |C| - 1 \text{ for other edge failure combinations}\}.
\end{aligned}$$

Therefore,

$$\mathbb{P}\{L_C = |C| - 1\} > \mathbb{P}\{L_v = |C| - 1\} \; \forall v \in C.$$

Similarly, it can be shown that

$$\mathbb{P}\{L_C = k\} > \mathbb{P}\{L_v = k\}\ \forall v \in C,\ \forall k \in \{0, 1, \ldots, |C| - 1\}. \tag{2.8}$$

Then the definition of CVaR and (2.8) imply the following:

$$CVaR_\alpha[L_C] > CVaR_\alpha[L_v]\ \forall v \in C. \tag{2.9}$$

Hence,

$$CVaR_\alpha[L_C] > \max_{v \in C} CVaR_\alpha[L_v].$$

□

The complexity of computing $\max_{v \in C} CVaR_\alpha[L_v]$ is $O(2^{|C|})$. If it exceeds the risk tolerance level K, then we can conclude that clique C is not robust. Otherwise, we proceed with computing or estimating CVaR value as described below:

- for all cliques C such that $|C| \leq 7$, we compute the value of $CVaR_\alpha[L_C]$ directly from (2.2) by explicit enumeration of the possible edge failures in C;
- for all cliques C such that $|C| > 7$, we use an estimation technique (2.6), with the number of generated scenarios equal to $|S| = 10m_C^2$, where m_C is the number of edges in the given clique C ($m_C = |C|(|C| - 1)/2$).

2.3.2 Remark on sampling approximation

First, let $F_\alpha(C, \zeta)$ denote the function characterizing $CVaR_\alpha[L_C(Y)]$:

$$F_\alpha(C, \zeta) = \zeta + \frac{1}{(1 - \alpha)} \mathbb{E}(L_C(Y) - \zeta)_+.$$

Then the sample-based estimate of $F_\alpha(C,\zeta)$ is

$$\bar{F}_\alpha(C,\zeta) = \zeta + \frac{1}{|S|(1-\alpha)} \sum_{s\in S} (L_C(Y^s) - \zeta)_+.$$

Let $Z = (L_C(Y) - \zeta)_+$ and $Z^s = (L_C(Y^s) - \zeta)_+$, then

$$F_\alpha(C,\zeta) = \zeta + \frac{1}{(1-\alpha)} \mathbb{E}[Z],$$

and

$$\bar{F}_\alpha(C,\zeta) = \zeta + \frac{1}{|S|(1-\alpha)} \sum_{s\in S} Z^s.$$

Next we will justify our choice of $|S| = 10m_C^2$, where $m_C = |C|(|C|-1)/2$, as a sufficient sample size such that $|\bar{F}_\alpha(C,\zeta) - F_\alpha(C,\zeta)|$ is small with high probability. To this aim, we will utilize an approach similar to the one described by Boginski et al. [9] and show that for any fixed $\epsilon > 0$ and $\beta \in (0,1]$, the sufficient sample size to guarantee that $\mathbb{P}\{|\bar{F}_\alpha(C,\zeta) - F_\alpha(C,\zeta)| \geq \epsilon\} \leq \beta$ is $|S| = O(n_C^2/\epsilon^2\beta^2)$, where $n_C = |C|$ is the number of vertices in a clique C.

To prove our statement it is sufficient to show that for any fixed $\epsilon > 0$ and $\beta \in (0,1]$ and $|S| = n_C^2/\epsilon^2\beta^2$, the following holds

$$\mathbb{P}\{|\frac{1}{|S|} \sum_{s\in S} Z^s - \mathbb{E}[Z]| \geq \epsilon\} \leq \beta. \tag{2.10}$$

Using Chebyshev's inequality we have

$$\mathbb{P}\{|\frac{1}{|S|} \sum_{s\in S} Z^s - \mathbb{E}[Z]| \geq \epsilon\} \leq \frac{\mathbb{V}ar^2[Z]}{\epsilon^2|S|} \tag{2.11}$$

Recall that our loss function is defined as $L_C(Y) = \max\limits_{i\in C} \sum\limits_{j\in C\setminus\{i\}} (1 - Y_{ij})$ and it is

bounded with values contained in the interval $[0, |C| - 1]$.

Then it is easy to see that

$$\mathbb{V}ar[Z] = \mathbb{V}ar[(L_C(Y) - \zeta)_+] \leq \mathbb{V}ar[L_C(Y)] \leq |C| - 1 \leq n_C - 1$$

Now (2.11) can be rewritten as

$$\mathbb{P}\{|\frac{1}{|S|}\sum_{s\in S} Z^s - \mathbb{E}[Z]| \geq \epsilon\} \leq \frac{(n_C - 1)^2}{\epsilon^2 |S|}, \tag{2.12}$$

and for (2.10) to hold we need $|S| \geq \lceil (n_C - 1)^2/\epsilon^2\beta^2 \rceil = O(n_C^2/\epsilon^2\beta^2)$.

In our approach, we set sample size $|S|$ equal to $10m_C^2 = 10(n_C(n_C-1)/2)^2$ which guarantees that

$$\mathbb{P}\{|\bar{F}_\alpha(C, \zeta) - F_\alpha(C, \zeta)| \geq \epsilon\} \leq \frac{2}{5n_C^2\epsilon^2}. \tag{2.13}$$

As we already mentioned, we use a sampling technique for all cliques with $n_C \geq 8$, so the probability bound derived in (2.13) might not seem very tight for smaller sizes of n_C. For example, for $n_C = 8$, the following holds t

However, to derive (2.13), we used the most trivial and rather weak bound on the variance of Z, $\mathbb{V}ar[Z] \leq n_C - 1$. Hence, a better bound on $\mathbb{V}ar[Z]$ will yield a smaller nominator value in the right-hand side of inequality (2.12) and thus a smaller probability for the same value of $|S|$.

Moreover, we performed a series of experiments to verify that the chosen sample size was indeed sufficient for accurate estimation of the value of $CVaR_\alpha[L_C(Y)]$. In these experiments, we considered cliques with $|C| = 8$ and $|C| = 9$. Also, for simplicity, we assumed the same edge failure probability in each tested clique. To generate edge failures we used two sample size values, $|S| = 10m^2$ and $|S| = 100m^2$. We used $\alpha = 0.9$ and ran 100 experiments for each combination of clique size, edge

failure probability and sample size; and in each of these 100 experiments we estimated the value of $CVaR_\alpha[L_C(Y)]$. As would be expected, results of these experiments indicate that the variance of an estimated value of $CVaR_\alpha[L_C(Y)]$ for $|S| = 100m^2$ is approximately 10 times smaller than that for $|S| = 10m^2$. However, with $|S| = 10m^2$ the variance is sufficiently small, specifically, on average, 0.013 percent of the sample mean. This provides evidence to support the assumption that $|S| = 10m^2$ is a sufficient sample size for an estimation of $CVaR_\alpha[L_C(Y)]$ within our heuristic approaches. In addition, we show in Section 2.7 that increasing the sample size in excess of $10m^2$ leads to a drastic increase of the computation time for the heuristics and the exact algorithm without any significant improvement in the accuracy of solutions as compared to that of the $10m^2$ sample size.

2.3.3 Robustness considerations during local search moves

Traditionally, the construction and local search-based heuristics for detecting cliques operate by performing addition, deletion, or swap of the vertices. For addition and swap operations, the next vertex is selected from the so called candidate set (and usually, but not always, the candidate set is represented by the vertices neighboring all the members of the solution). If there is more than one vertex in the candidate set, different selection rules can be exploited, such as greedy, random, or other problem-specific ones. For detection of a robust clique, given the current solution C and the candidate set D, we have to consider the following when selecting the vertices:

(1) If C is robust, then which vertex from D should be added to solution C, so that C is likely to stay robust?

(2) If C is not robust, then which vertex (vertices) in C should be deleted (or swapped with a vertex from D) so that C is likely to become robust?

It is computationally challenging to evaluate CVaR for every neighboring solution obtained by the addition (deletion or swap) of each candidate vertex. Hence, we propose to make a selection based on the estimation of a vertex contribution to the value of CVaR. An intuitive approach is to compute the probability of some of the largest losses, since CVaR is defined as the average of such losses. The largest loss associated with each vertex v with respect to solution C is the failure of all the edges between vertex v and the rest of the vertices in C. Then the probability of such loss is,

$$P_C(v) = \prod_{i \in C \setminus \{v\}} p_{iv}, \tag{2.14}$$

where p_{iv} is the failure probability of edge (i, v).

If we want to expand the solution by moving a vertex v from D to C, naturally, we would like to pick the vertex that would likely lead to the smallest possible value of $CVaR_\alpha[L_{C'}]$, where $C' = C \cup \{v\}$. So, it makes sense to select a vertex with the smallest value of $P_C(v)$ among all $v \in D$, and we call such vertex the *most robust-favorable*.

Definition 2. *A vertex v^* is the most robust-favorable among the vertices in the candidate set D if*

$$P_C(v^*) = \min_{v \in D} \{P_C(v)\}. \tag{2.15}$$

Similarly, if we want to remove a vertex from C (i.e., when C is not robust) in order to obtain a clique with the smaller value of CVaR, we would like to remove a vertex that contributes the most to $CVaR_\alpha[L_C]$. Such vertex is the one with the largest $P_C(v)$, and will be referred to as the *least robust-favorable*.

Definition 3. *A vertex v^* is the least robust-favorable among the vertices in the*

solution set C if

$$P_C(v^*) = \max_{v \in C}\{P_C(v)\}. \tag{2.16}$$

For swap operations, we pick $(u, v) \in C \times D$ that yields the largest difference between $P_C(u)$ and $P_{C \cup \{v\} \setminus \{u\}}(v)$.

Even though adding the most robust-favorable vertex, removing the least-favorable vertex, or swapping two vertices as just described can possibly result in a solution that is not robust, this would indicate that a robust solution sought is unlikely to be found in the current neighborhood, thus encouraging diversification of the search.

2.3.4 Bounds on the solution size

For all the developed heuristics, we assume that the size of the largest deterministic clique (ω) is given. If it is not known in advance, it can be computed using one of the many existing effective exact algorithms [58, 72, 15, 18, 6, 70]. Once known, it can be used to establish an upper bound on the size of the largest robust clique. Moreover, based on the smallest (p_{min}) and the largest (p_{max}) edge failure probabilities in the graph, the upper ($\omega_{p_{min}}$) and lower ($\omega_{p_{max}}$) bounds on the solution size can be extracted from the tables in Appendix A. Then the overall lower bound representing the size of the guaranteed feasible *robust* solution, and the upper bound used as one of the criteria for algorithm termination, are:

$$\omega_{LB} = \min\{\omega, \omega_{p_{max}}\} \text{ and } \omega_{UB} = \min\{\omega, \omega_{p_{min}}\},$$

respectively.

These bounds are determined and used during the initialization step in all the algorithms presented in this chapter.

2.4 TABU-risk algorithm

TABU-risk algorithm presented in this section is an adaptation of TS for the detection of robust cliques. As in many TS algorithms for the maximum clique problem, in TABU-risk the search space consists of cliques in the graph. The neighborhood of a solution is defined using three operators that add, swap and drop vertices (see Section 2.4.2).

2.4.1 Initialization

After the lower bound ω_{LB} is determined, we attempt to construct an initial clique of size at most $(\omega_{LB}+1)$ using a randomized heuristic. This heuristic picks a vertex at random from a candidate set and places it in the solution. In the very beginning the candidate set comprises all the vertices with degree at least $(\omega_{LB}-1)$. Every time a vertex is added to the solution, the candidate set is updated and only contains the vertices neighboring all those already in the solution. The heuristic keeps executing until it constructs a clique of size $(\omega_{LB}+1)$ or the candidate list is empty. In practice, ω_{LB} is usually considerably small, therefore, the clique of size $(\omega_{LB}+1)$ can be easily found. Recall that robustness is guaranteed for any clique of size ω_{LB}, but not for larger sizes. Hence, we start with a clique of the smallest size for which robustness is not trivial and needs to be ensured.

2.4.2 Move operators and neighborhoods

Given a graph $G=(V,E)$ and a feasible solution $C \subset V$, the neighborhood $\mathcal{N}(C)$ is defined as a set of solutions that can be obtained from C using a certain move operator mv:

$$\mathcal{N}(C) = \{C' : C' = C \oplus mv\}.$$

Different choices of the move operator mv result in different neighborhood structures. Next, we define three move operators, *ADD, SWAP*, and *DROP*.

For $C \subset V$, let $N_G^{\cap}(C) = \{i \in V \setminus C : (i, v) \in E, \forall v \in C\}$ be the set of vertices that are adjacent to each vertex in C. Then the three move operators are defined as follows.

$ADD(v)$**:** Add a vertex $v \in N_G^{\cap}(C)$ to the current solution C. The neighborhood associated with this move is

$$\mathcal{N}_A(C) = \{C' : C' = C \oplus ADD(v), v \in N_G^{\cap}(C)\}.$$

This move requires calculation of $P_C(v)$ for all $v \in N_G^{\cap}(C)$. It takes $O(|C|)$ to calculate $P_C(v)$ for each $v \in N_G^{\cap}(C)$, where size of $N_G^{\cap}(C)$ is $O(n)$, hence yielding overall move's complexity of $O(|C|n)$.

$SWAP(u, v)$**:** Drop a vertex u from the current solution C and add a vertex $v \in N_G^{\cap}(C \setminus \{u\})$ to the solution. The neighborhood associated with this move is

$$\mathcal{N}_S(C) = \{C' : C' = C \oplus SWAP(u, v), u \in C, v \in N_G^{\cap}(C \setminus \{u\})\}.$$

It takes $O(|C|)$ to evaluate each difference $P_C(u) - P_{C\cup\{v\}\setminus\{u\}}(v)$, where $(u, v) \in C \times N_G^{\cap}(C \setminus \{u\})$. Since $|C \times N_G^{\cap}(C \setminus \{u\})| = O(|C|n)$, the overall complexity of a swap move is $O(|C|^2 n)$.

$DROP(u)$**:** Drop a vertex u from the current solution C. The neighborhood associated with this move is defined as

$$\mathcal{N}_D(C) = \{C' : C' = C \oplus DROP(u), u \in C\}.$$

Similarly, to perform $DROP(u)$, it is required to calculate $P_C(v)$ for all $u \in C$, which yields move's complexity of $O(|C|^2)$.

2.4.3 Tabu list

To diversify the search, TABU-risk uses a *tabu list* that keeps historical information about the vertices' migration in order to avoid repeated usage of the same vertices in future moves. Following the approach proposed by Wu et al. for the maximum weight clique problem [73], we use a single tabu list consisting of a random number of vertices previously dropped from the solution. Namely, a vertex v dropped from the solution is added to the tabu list and is kept there for the next $T(v)$ iterations, where T, referred to as *tabu tenure*, is defined as follows:

$$T(v) = \begin{cases} T_1 + rand(B), & \text{for vertices removed by } SWAP \\ T_1, & \text{for vertices removed by } DROP, \end{cases}$$

where T_1 is some constant ($T_1 = 7$ is used in our experiments, as suggested in [73]), B is the size of $N_G^{\cap}(C)$ with C being the current solution, and $rand(B)$ is a uniformly distributed random integer in the range between 0 and B. The vertex is called *tabu* if it is in the tabu list, and *non-tabu* – otherwise.

2.4.4 The procedure

This section describes how the TABU-risk algorithm exploits the move operators introduced above in conjunction with tabu rules in order to explore the solution space. The outline of the proposed approach is presented in Algorithm 1.

During the execution of the algorithm's steps, we maintain a *candidate* clique C that is first tested for robustness.

If the CVaR requirements are satisfied we proceed as follows. If $|C|$ exceeds

Algorithm 1 TABU-risk algorithm for detecting robust clique

Determine ω_{LB}, ω_{UB}
Construct a clique C of size $\omega_{LB}+1$ (as described in Section 2.4.1)
$Iter \leftarrow 0, \omega \leftarrow 0$
while $Iter \leq L$ **do**
  $Iter \leftarrow Iter + 1$
  if C satisfies CVaR constraints **then**
    if $|C| > \omega$ **then**
      $\omega \leftarrow |C|$
      $C^* \leftarrow C$
      $Iter \leftarrow 0$
      if $\omega = \omega_{UB}$ **then**
        return C^*
    else
      $Iter \leftarrow Iter + 1$
    if there is a non-tabu vertex in $N_G^{\cap}(C)$ **then**
      $C \leftarrow C \oplus ADD(v)$, where v is the most robust-favorable candidate (2.15)
    else
      if there is a pair (u, v), $u \in C$ and v is a non-tabu vertex in $N_G^{\cap}(C \setminus \{u\})$ **then**
        $C \leftarrow C \oplus SWAP(u, v)$, where (u, v) maximizes $P_C(u) - P_{C\cup\{v\}\setminus\{u\}}(v)$
      else
        $C \leftarrow C \oplus DROP(u)$, where u is the least robust-favorable candidate
  else
    if there is a pair $(u, v), u \in C$ and v is a non-tabu vertex in $N_G^{\cap}(C \setminus \{u\})$ **then**
      $C \leftarrow C \oplus SWAP(u, v)$, where (u, v) maximizes $P_C(u) - P_{C\cup\{v\}\setminus\{u\}}(v)$
    else
      $C \leftarrow C \oplus DROP(u)$, where u is the least robust-favorable candidate
return C^*

the current lower bound ω, then the solution is recorded as C^*, the best solution found so far. If this is the case, the variable $Iter$, which counts the number of consecutive iterations without improvement, is reset to 0. Next, we explore the local neighborhoods of C in the following way. First, we attempt to expand C by adding a new vertex v, which is the most robust-favorable (as described in Section 2.3.3) non-tabu vertex in $N_G^{\cap}(C)$. If there is no such qualified vertex, the algorithm proceeds to the next step, which attempts to swap the most robust-favorable and non-tabu pair of vertices (u, v), which is a pair of vertices that maximizes $P_C(u) - P_{C\cup\{v\}\setminus\{u\}}(v)$. For diversification purposes, the move is performed even if this difference is negative. If there is no qualified pair of vertices, we still diversify the search by dropping the least robust-favorable vertex from C.

Alternatively, if C does not satisfy the CVaR requirements, we follow a different path. Namely, instead of increasing the size of the solution, the aim is to make it robust. First, we try to achieve this objective by finding a clique C' obtained from C by performing a non-tabu swap. If this attempt fails, we proceed to dropping the least robust-favorable vertex from C.

The algorithm repeats as long as $Iter$ does not exceed a given threshold L and the solution is less than ω_{UB}.

2.5 STABULUS-risk algorithm

The metaheuristic presented here borrows main features from TS-based algorithm for detecting stable sets called STABULUS [30]. This method performs a local search on infeasible solutions. A similar approach for the maximum clique problem was proposed in [74]. The local search in this case is performed on the sets of a fixed size k that are not cliques, by making swaps leading to the increase of the number of edges in the corresponding induced subgraph. When the number of edges is

maximum possible $(k(k-1)/2)$, the feasible solution – a clique of size k is found, and the algorithm proceeds by incrementing the value of k.

Algorithm 2 Multistart STABULUS-risk algorithm for detecting robust clique

```
Determine ω_LB, ω_UB
k ← ω_LB + 1
Construct an initial solution C of size k (as described in Section 2.5.1)
Restart ← 0
while (k ≤ ω_UB) AND (Restart ≤ RestartMax) do
    Restart ← Restart + 1
    if f(C) < k(k − 1)/2 then
        C ← PhaseOne(C, L_1)
    if f(C) = k(k − 1)/2 then
        (C, check) ← PhaseTwo(C, L_2)
        if check = true then
            k ← k + 1
            C^best ← C
            Restart ← 0
            Pick v ∈ V \ C with the maximum number of neighbors in C
            C ← C ∪ {v}
        else
            Construct a solution C of size k (as described in Section 2.5.1)
    else
        Construct a solution C of size k (as described in Section 2.5.1)
return C^best
```

To adapt STABULUS to the robust version of the maximum clique problem, we propose a two-phase algorithm. In Phase I the algorithm works as the deterministic one (with slight alterations described in Section 2.5.3), trying to find a clique of size k. Successful termination of the first phase does not guarantee that a robust solution is found. If the solution does not meet the risk requirements, Phase II is initiated in an attempt of finding a robust solution in the local neighborhood of the current solution (Section 2.5.4). The algorithm restarts with $k = k+1$ if the result of Phase

II is successful, or with the same k, otherwise. The outline of STABULUS-risk is presented in Algorithm 2.

2.5.1 Initialization and restart

The initial solution C is constructed as a set of vertices of size $k = \omega_{LB} + 1$. It does not necessarily have to be a clique, but it is found using a clique construction heuristic, similar to one applied in Section 2.4.1. The very first vertex v_1 to be added to the solution is picked randomly from the set of vertices having degree at least $(k-2)$. Then the candidate list is constructed from the neighbors of v_1 in G. All the subsequent selections of the vertices from the candidate list are also random. After each vertex addition the candidate list is updated by removing all non-neighbors of the current solution. These steps are executed until a clique of size k is constructed. If the candidate list becomes empty before the solution size reaches k, we add $(k-|C|)$ vertices from set $V \setminus C$. The vertices selected in this situation are those with the largest number of neighbors in C.

The same heuristic is used for constructing solutions during the restarts occurring when: (1) Phase I was not successful (could not find a clique), or (2) Phase II was not successful (could not find robust clique).

If Phase II yields a robust clique, then k is incremented by 1, and the current solution is expanded by one extra vertex from set $V \setminus C$. The vertex selected is such that it has the largest number of neighbors in the current solution C.

2.5.2 Tabu lists and tenures

Every vertex being added to or dropped from the solution is included in the respective tabu list.

For Phase I we have the tabu tenures, similar to [74]:

$$T(v) = \lfloor 0.6l \rfloor + rand(4), \text{ for any added vertex } v,$$

and

$$T(u) = l + rand(6), \text{ for any removed vertex } u,$$

where $l = \min\{10, l_1\}$ and l_1 is the number of missing edges in the induced subgraph corresponding to the current solution; and $rand(b)$ is a random integer in the range between 0 and b.

In Phase II the following tabu tenures were used:

$$T(v) = 4 + rand(2), \text{ for any added vertex } v,$$

$$T(u) = 7 + rand(4), \text{ for each removed vertex } u.$$

The pair of vertices (u, v) is tabu if both of the vertices are still in the tabu list.

2.5.3 Phase I

In Phase I of STABULUS-risk the quality of the solution C is evaluated by the number of edges in $G[C]$, denoted by $f(C)$. The optimal value of $f(C)$ is $k(k-1)/2$, where $k = |C|$. Once it is reached, Phase I terminates. It also terminates if the number of non-improving iterations has reached the maximum allowed value L_1. The outline of Phase I is presented in Algorithm 3.

The solution neighborhood structure used in local search of Phase I is the following:

$$\mathcal{N}(C) = \{C' : C' = C \oplus SWAP(u, v), u \in D^{min}, v \in D^{max}\},$$

where $D^{min} \subset C$ contains all the vertices $u \in C$ with the minimum degree $d_{G[C]}(u)$

in the subgraph $G[C]$:

$$D^{min} = \{u' : u' \in C, d_{G[C]}(u') = \min_{u \in C}\{d_{G[C]}(u)\}\},$$

and $D^{max} \subset V \setminus C$ consists of all the vertices $v \in V \setminus C$ with the maximum degree in the subgraph $G[C \cup \{v\}]$:

$$D^{max} = \{v' : v' \in V \setminus C, d_{G[C\cup\{v'\}]}(v') = \max_{v \in V \setminus C}\{d_{G[C\cup\{v\}]}(v)\}\}.$$

If there are more than one vertex in each set D^{min} and D^{max}, then the pair (u, v) with the largest value of $f(C \setminus \{u\} \cup \{v\}) - f(C)$ (which may be negative) is selected.

Algorithm 3 Phase I of STABULUS-risk algorithm

```
procedure PhaseOne(C, L_1)
    Iter_1 ← 0, C* = C
    while Iter_1 ≤ L_1 do
        check ← false
        for each (u, v) such that u ∈ D^min, v ∈ D^max do
            if ((u, v) is non-tabu) OR (f(C ∪ {v} \ {u}) > f(C*)) then
                δ(u, v) ← f(C ∪ {v} \ {u}) − f(C)
                check ← true
        if check = true then
            Pick (u, v) with max{δ(u, v)}
            if ties then pick (u, v) with max{P_C(u) − P_{C∪{v}\{u}}(v)}
        else
            Pick vertices u ∈ C and v ∈ V \ C at random
        C ← C ⊕ SWAP(u, v)
        if f(C) > f(C*) then
            C* ← C
            Iter_1 ← 0
        else
            Iter_1 ← Iter_1 + 1
    return C*
```

If in any pair (u, v) both vertices are tabu, they are only considered in selection process if $f(C \setminus \{u\} \cup \{v\}) > f(C^*)$ (*aspiration criterion*), where $f(C^*)$ is the value of the best solution found so far. If there is still more than one candidate for a swap, then the ties are broken using the robustness criterion as follows. A pair with the largest value of $P_C(u) - P_{C\setminus\{u\}}(v)$ is selected.

If all the pairs were tabu and did not satisfy aspiration criterion, then the pair for a swap is selected randomly. When the swap is performed, both vertices are added to the corresponding tabu lists (see Section 2.5.2).

2.5.4 Phase II

Phase II of STABULUS-risk (see Algorithm 4) attempts to make the current solution robust by performing a local search in the neighborhood of this solution. For this purpose we used neighborhood $\mathcal{N}_S$ described in Section 4.2. However, instead of exploring the entire neighborhood to detect the most robust-favorable pair of vertices for a swap, we select the first found non-tabu pair of vertices (u, v) such that

Algorithm 4 Phase II of STABULUS-risk algorithm

```
procedure PhaseTwo(C, L_2)
    if C satisfies CVaR constraints then
        return C, true
    else
        Iter_2 ← 0
        while (Iter_2 ≤ L_2) do
            Iter_2 ← Iter_2 + 1
            Find (u, v) such that u ∈ C, v ∈ N_G^∩(C)
            if ((u, v)) is non-tabu) AND (P_C(u) > P_{C∪{v}\{u}}(v)) then
                C ← C ⊕ SWAP(u, v)
        if C satisfies CVaR constraints then
            return C, true
        else
            return ∅, false
```

$P_C(u) > P_{C\cup\{v\}\setminus\{u\}}(v)$ and update the solution. Here, non-tabu status applies if at least one of the vertices is non-tabu. We perform these steps as long as we can find a non-tabu pair (u, v) with $P_C(u) > P_{C\cup\{v\}\setminus\{u\}}(v)$ and the number of the steps is less than L_2. Then it is verified whether the solution satisfies CVaR constraints, and Phase II either outputs a *true* value for *check* argument along with the solution or terminates with a *false* value for *check*.

2.6 GRASP-risk algorithm

In this section we present GRASP-risk algorithm, the modification of the GRASP approach for the maximum clique problem [3]. Similarly to its original counterpart, GRASP-risk algorithm consists of two phases: (1) construction of the initial solution (a clique, possibly infeasible in terms of risk constraints, i.e., not robust), and (2) performing a local search in an attempt to improve the initial solution (by increasing its size), or, if it is infeasible (not robust), make it feasible (robust). The algorithm restarts a certain number of times and outputs the best found solution. The outline of the method is presented in Algorithm 5.

Algorithm 5 GRASP-risk for detecting robust cliques

```
Determine ω_LB, ω_UB
Restart ← 0
ω ← 0
while (ω < ω_UB) AND (Restart ≤ RestartMax) do
    Restart ← Restart + 1
    C ← GreedyRandomizedConstruction (as described in Section 2.6.1)
    C ← LocalSearch(C) (as described in Section 2.6.2)
    if |C| > ω then
        ω ← |C|
        C* ← C
return ω, C*
```

2.6.1 Construction of the solution

Construction of an initial solution in GRASP is implemented using the so called restricted candidate list (RCL). Let $D = N_G^{\cap}(C)$ be the candidate set. For GRASP-risk approach we construct the RCL in the following way. First, let RCL_d be the list of the best candidates with respect to the size criterion, and let RCL_p be the list containing the best candidates in terms of the risk criterion. Then, we construct RCL by finding an intersection of RCL_d and RCL_p.

We have:

$$d_{G[D]}^{max} = \max\{d_{G[D]}(v) : \forall v \in D\},\ d_{G[D]}^{min} = \min\{d_{G[D]}(v) : \forall v \in D\},$$

$$P_C^{max} = \max\{P_{C\cup\{v\}}(v) : \forall v \in D\},\ P_C^{min} = \min\{P_{C\cup\{v\}}(v) : \forall v \in D\}.$$

Let τ_d and τ_p define the thresholds such that

$$\tau_d = d_{G[D]}^{min} + \alpha(d_{G[D]}^{max} - d_{G[D]}^{min}),$$

$$\tau_p = P_C^{min} + (1-\alpha)(P_C^{max} - P_C^{min}),$$

where $\alpha \in [0,1]$. Then

$$\mathrm{RCL}_d = \{v \in D : d_{G[D]}(v) \geq \tau_d\},$$

$$\mathrm{RCL}_p = \{v \in D : P_C(v) \leq \tau_p\},$$

and

$$\mathrm{RCL} = \mathrm{RCL}_d \cap \mathrm{RCL}_p.$$

If for a certain value of α, set RCL is empty, then the value of α is decreased (by 0.05) iteratively until RCL contains at least one vertex. In the worst-case scenario, when $\alpha = 0$, then $\tau_d = d^{min}_{G[D]}, \tau_p = P^{max}_C$, RCL_d and RCL_p both contain all the vertices from D, therefore their intersection is nonempty.

When RCL is not empty, a vertex from this set is picked randomly and added to the solution (see Algorithm 6). Then the candidates set is updated, and the same procedure of constructing RCL is repeated until the candidate set D is empty or the size of the clique is equal to the upper bound ω_{UB}. Note that the solution might be infeasible in terms of the CVaR constraint (not robust), but it will get adjusted during the local search procedure (Section 2.6.2).

Algorithm 6 Construction phase of GRASP-risk

```
procedure GreedyRandomizedConstruction(ω_UB)
    C ← ∅, D ← ∅
    Pick randomly v ∈ V
    C ← C ∪ {v}
    D ← D ∪ {i : (v, i) ∈ E}
    while (|C| < ω_UB) AND (D ≠ ∅) do
        RCL ← ConstructRCL(D)
        Pick randomly v ∈ RCL
        C ← C ∪ {v}
        D ← D ∩ {i : (v, i) ∈ E}
    return S
```

2.6.2 Local search procedure

After the initial solution is constructed, and depending on whether it is robust or not, different local search techniques will be utilized. If it is robust, then the local search referred to as (1,2)-exchange in [3] will be applied in an attempt to expand the current solution in size. In case the solution does not satisfy the risk constraints,

its local neighborhood defined by the operators $SWAP(u, v)$ and $DROP(u)$ (if necessary) will be explored.

The (1,2)-exchange local search explores the local neighborhood defined as:

$$\mathcal{N}_E(C) = \{C' : C' = C \oplus EXCH(u; v_1, v_2), u \in C, v_1, v_2 \in N_G^{\cap}(C \setminus \{u\}), (v_1, v_2) \in E\},$$

where $EXCH(u; v_1, v_2)$ is the move operator that drops a vertex u from the current solution C and adds two adjacent vertices $v_1, v_2 \in N_G^{\cap}(C \setminus \{u\})$ to the solution. Note that the size of $\mathcal{N}_E(C)$ is $O(|C|n^2)$. We employ the first improvement strategy and move to the first robust neighbor C' found in $\mathcal{N}_E(C)$. If $\mathcal{N}_E(C)$ is empty or contains no robust solutions, the local search terminates and returns the current solution.

If the constructed solution C is not robust, the neighborhood $\mathcal{N}_S(C)$ is explored as follows. For each $u \in C$ we seek the most robust-favorable vertex $v \in N_G^{\cap}(C \setminus \{u\})$ until we find a robust solution $C \oplus SWAP(u, v)$, at which point we perform the swap and attempt (1,2)-exchange. If no robust solution is found in $\mathcal{N}_S(C)$, the move $DROP(u)$, where u is the least robust-favorable vertex, is iteratively applied to the current solution until it becomes robust.

The outline of the local search procedure is presented in Algorithm 7.

2.7 Computational experiments

2.7.1 Experiment instances and settings

To test the performance of the proposed heuristics we used two sets of graphs instances with the number of nodes ranging from 105 to 4941.

The first set of graph instances are the real-life graphs taken from the 10^{th} DIMACS Implementation Challenge [23]. As most of the real-life networks, these graphs are sparse and have rather small clique number. The second set of graph instances

Algorithm 7 Local search phase of GRASP-risk

```
procedure LocalSearch(C)
    if C satisfies CVaR constraints then
        Starting with C, find C* with no robust solutions in N_E(C*)
    else
        for each u ∈ C do
            Find the most robust-favorable vertex v ∈ N_G^∩(C \ {u})
            if C ⊕ SWAP(u, v) is robust then
                C ← C ⊕ SWAP(u, v)
                Starting with C, find C* with no robust solutions in N_E(C*)
                return C*
        while C does not satisfy CVaR constraints do
            C ← C ⊕ DROP(u), where u is the least robust-favorable candidate
        C* = C
    return C*
```

was taken from the 2nd DIMACS Implementation Challenge [22]. Most of these graphs are very dense and have larger clique number than those from the first set.

For the experiment purposes the edge failure probabilities were assigned randomly to one of the values used in the tables in Appendix A ($p = 0.05, 0.075, \ldots, 0.2$). The confidence level α for CVaR used in the experiments was set to 0.9.

To test the proposed heuristics on the given instances, the following settings were used:

TABU-risk: The search depth, L, is set to 100.

STABULUS-risk: The number of restarts without improvement in the solution size, $RestartMax$, is set to $|V|$ if $|V| < 25$, and to $\lfloor \frac{|V|}{\omega_{LB}} \rfloor$, otherwise. The search depths for Phase I, L_1, and Phase II, L_2, are set to 50, and 10, respectively.

GRASP-risk: The number of multiple starts, $RestartMax$ is equal to $|V|$ if $|V| < 25$, and to $\lfloor \frac{|V|}{\omega_{UB}} \rfloor$, otherwise.

The search depth parameters were chosen according to the suggestions from [73, 74] and were tuned based on the results of the preliminary experiments. The number of restarts for STABULUS-risk and GRASP was selected to be roughly proportional to $|V|/\omega$, which is a lower bound on the clique partitioning number, to ensure a minimally sufficient level of diversification of search. Lower and upper bound on ω was used for STABULUS-risk and GRASP, respectively, based on the preliminary experiments.

The heuristics were coded in C++ and compiled using Microsoft Visual Studio 2010 on a PC with 2.40 GHZ CPU and 12 GB RAM. Estimation of CVaR value was implemented using GUROBI solver. Since all of the heuristics presented in this chapter use randomness to some extent, they were run 50 times, each time with a different random seed.

We also compared the performance of the heuristics to that of a modification of Östergård's algorithm [58]. The modification consisted in execution of the CVaR verification procedure (Section 2.3.1) every time a vertex is added to a candidate set.

2.7.2 Discussion of the results

The experiments were conducted for two values of the robustness tolerance level K (see Definition 1), 3 and 4, and the results are reported in Tables 2.1–2.4.

The first four columns in Tables 2.1 and 2.2 contain general information about the particular graph instance: name, number of vertices ($|V|$), number of edges ($|E|$), and the size of the largest deterministic clique (ω). The rest of the columns report the sizes of the robust cliques obtained using each heuristic, as well as the exact algorithm. In particular, for heuristics, the average ($\overline{\omega_r}$) and the best (ω_r^{max}) solutions obtained in 50 runs are reported. If for a certain instance the size of the lower bound (ω_{LB}) equals to the size of the largest deterministic clique (ω), the

Table 2.1: Robust cliques obtained by the metaheuristics (50 runs) and the exact algorithm with $CVaR_{0.9}[L_C] \leq 3$, $|S| = 10m_C^2$, and the corresponding bounds $\omega_{p_{max}} = 4, \omega_{p_{min}} = 10$.

	$\|V\|$	$\|E\|$	ω	TABU		STABULUS		GRASP		Exact
				$\overline{\omega_r}$	ω_r^{max}	$\overline{\omega_r}$	ω_r^{max}	$\overline{\omega_r}$	ω_r^{max}	ω_r
polbooks	105	441	6	5.62	6	6	6	5.88	6	6*
football	115	613	9	5.84	6	6	6	6	6	6*
celeg._metab.	453	2025	9	6	6	6	6	6	6	6*
email	1133	5451	12	6.02	7	6.98	7	6.74	7	7*
power	4941	6594	6	4.04	6	6	6	5.86	6	6*
brock200_4	200	13089	17	7.12	8	7.14	8	7	7	7
brock400_2	400	59786	29	8	8	7.72	8	7.02	8	7
brock400_4	400	59765	33	8	8	7.8	8	7	7	7
brock800_2	800	208166	24	8	8	7.98	8	7	7	7
brock800_4	800	207643	26	8	8	8	8	7	7	7
C125.9	125	6963	34	7.36	8	7.48	8	7	7	7
C250.9	250	27984	44	7.98	8	7.64	8	7.02	8	7
C500.9	500	112332	57	8	8	7.8	8	7	7	7
C1000.9	1000	450079	68	8	8	8	9	7	7	7
C2000.5	2000	999836	16	7.96	8	8	8	7	7	7
C2000.9	2000	1799532	80	8.16	9	8.02	9	7	7	7
dsjc500_5	500	62624	13	7	7	7	7	7	7	7
dsjc1000_5	1000	249826	15	7.24	8	7.5	8	7	7	7
MANN_a27	378	70551	126	7.54	8	7.54	8	7	7	7
MANN_a45	1035	533115	345	7.92	8	7.96	9	7	7	7

heuristic stops at the initialization phase and yields the largest deterministic clique as a solution. For such instances the values of $\overline{\omega_r}$ and ω_r^{max} are marked with an asterisk.

For the exact algorithm, the cardinality of the robust clique found (denoted by ω_r) for each case is reported in the last column of Tables 2.1 and 2.2. The running time limit was set to 7200 seconds, however, the actual running time of the algorithm was larger to allow the completion of the current iteration. If an optimal solution was not produced within the corresponding time limit, we report the best solution

Table 2.2: Robust cliques obtained by the metaheuristics (50 runs) and the exact algorithm with $CVaR_{0.9}[L_C] \leq 4$, $|S| = 10m_C^2$, and the corresponding bounds $\omega_{p_{max}} = 6, \omega_{p_{min}} = 16$.

	$\|V\|$	$\|E\|$	ω	TABU		STABULUS		GRASP		Exact
				$\overline{\omega_r}$	ω_r^{max}	$\overline{\omega_r}$	ω_r^{max}	$\overline{\omega_r}$	ω_r^{max}	ω_r
polbooks	105	441	6	6*	6*	6*	6*	6*	6*	6*
football	115	613	9	7.4	9	8.7	9	8.52	9	9*
celeg._metab.	453	2025	9	8.96	9	9	9	9	9	9*
email	1133	5451	12	7.22	9	9	9	8.92	9	9*
power	4941	6594	6	6*	6*	6*	6*	6*	6*	6*
brock200_4	200	13089	17	10	10	10	10	9.8	10	10
brock400_2	400	59786	29	11	11	10.56	11	10	10	9
brock400_4	400	59765	33	11	11	10.5	11	10	10	9
brock800_2	800	208166	24	10.98	11	10.76	11	10	10	9
brock800_4	800	207643	26	10.9	11	10.68	11	10	10	9
C125.9	125	6963	34	10.76	11	10.58	11	10	10	9
C250.9	250	27984	44	11	11	10.3	11	10	10	9
C500.9	500	112332	57	11.08	12	10.52	11	10.02	11	10
C1000.9	1000	450079	68	11.82	12	10.72	11	10.02	11	9
C2000.5	2000	999836	16	10.04	11	10.08	11	10	10	9
C2000.9	2000	1799532	80	12	12	10.94	12	10.04	11	10
dsjc500_5	500	62624	13	9.54	10	9.84	10	9.22	10	9
dsjc1000_5	1000	249826	15	10	10	10	10	9.9	10	9
MANN_a27	378	70551	126	10.96	11	10.52	11	10	10	10
MANN_a45	1035	533115	345	11	11	10.58	12	10.02	11	10

found within the limit. If an optimal solution was obtained, it is marked with an asterisk in Tables 2.1 and 2.2.

Tables 2.3 and 2.4 report the average (over the 50 runs) running time for all three heuristics and the time taken by the exact algorithm. For each heuristic, the average time it took to reach the best found solution for the first time during each run ($\overline{t^*}$) and the average total time of algorithm execution ($\overline{T}$) are presented. For the exact algorithm, the time it took to reach the best found solution for the first time ($t*$) and the total time of the algorithm (T) are reported. For most of the considered

graphs, $\overline{t^*}$ was considerably smaller than $\overline{T}$ for heuristics and t^* was considerably smaller than T for the exact algorithm, meaning that the high-quality solutions are typically computed fast. For the instances, for which $\omega_{LB} = \omega$, the running time is not reported.

Table 2.3: Running time comparison of the metaheuristics and the exact algorithm ($CVaR_{0.9}[L_C] \leq 3$ and $|S| = 10m_C^2$)

	TABU		STABULUS		GRASP		Exact	
	$\overline{t^*}$	$\overline{T}$	$\overline{t^*}$	$\overline{T}$	$\overline{t^*}$	$\overline{T}$	t^*	T
polbooks	<0.01	0.01	0.03	0.03	0.03	0.12	0.03	0.04
football	<0.01	5.34	0.01	13.35	2.31	16.14	0.01	39.89
celeg._metab.	<0.01	28.29	<0.01	42.13	2.92	55.38	0.01	105.17
email	0.14	6.2	8.16	11.58	13.63	39.28	82.01	205.59
power	< 0.01	0.03	2.6	2.6	0.05	0.14	0.36	0.39
brock200_4	4.71	62.27	2.54	33.05	6.98	48.49	192.25	8473.67
brock400_2	7.99	152.22	10.33	130.48	5.34	68.98	45.40	7540.08
brock400_4	10.76	154.88	11.97	142.22	5.78	67.75	183.80	7255.16
brock800_2	6.72	153.32	17.82	316.57	5.34	133.65	31.64	7375.88
brock800_4	9.91	157.51	21.45	346.12	6.68	132.8	313.98	8578.13
C125.9	10.73	95.68	3.11	35.81	4.22	22.95	52.10	9548.19
C250.9	11.21	176.33	4.61	71.24	6.61	38.72	115.19	7438.79
C500.9	3.36	154.61	10.35	169.14	7.16	79.61	30.87	8653.11
C1000.9	2.65	161.52	31.12	399.24	6.36	155.68	368.94	9278.53
C2000.5	14.47	154.58	31.36	818.29	6.28	396.18	56.19	8024.46
C2000.9	16.95	171.02	24.66	766.66	6.97	345.69	26.05	7515.24
dsjc500_5	2.29	32.28	1.29	41.71	14.88	150.93	0.81	7208.39
dsjc1000_5	5.22	73.05	27.41	266.51	13.17	311.44	22.45	7546.67
MANN_a27	6.32	108.1	6.44	121.25	5.59	59.18	71.72	7704.49
MANN_a45	4.64	146.38	15.87	378.12	5.07	159.78	89.75	15742.2

As it can be seen from the tables, the heuristics outperform the exact algorithm in terms of total running time, and, in many cases, even in terms of the time when the best found solution was detected for the first time. The heuristics also dominate

Table 2.4: Running time comparison of the metaheuristics and the exact algorithm ($CVaR_{0.9}[L_C] \leq 4$ and $|S| = 10m_C^2$)

	TABU		STABULUS		GRASP		Exact	
	$\overline{t^*}$	$\overline{T}$	$\overline{t^*}$	$\overline{T}$	$\overline{t^*}$	$\overline{T}$	t^*	T
polbooks	-	-	-	-	-	-	-	-
football	0.96	4.87	3.58	5.45	9.41	22.75	6.04	6.57
celeg._metab.	6.52	6.58	3.68	3.68	7	33.87	6.04	6.54
email	0.89	13.69	3.37	8.43	7.21	23.73	6.17	325.59
power	-	-	-	-	-	-	-	-
brock200_4	7.75	242.59	4.66	150.71	68.51	213.95	1879.86	12450.4
brock400_2	20.45	98.81	28	164.58	13.97	237.41	22.74	9464.29
brock400_4	31.91	86.95	49.48	204.4	17.48	233.88	182.57	10778.3
brock800_2	46.89	79.04	165.51	318.9	60.73	759.16	26.57	7214.69
brock800_4	49.02	81.15	139.2	338.11	43.93	775.67	162.60	11441.5
C125.9	36.1	108.11	16.42	58.67	10.2	71.01	6.64	9414.29
C250.9	13.69	146.16	19.09	156.68	12.66	126.4	24.95	14240.3
C500.9	16.09	299.88	46.84	242.98	15.86	261.43	4589.14	14553.3
C1000.9	50.48	890.51	108.02	339.89	24.01	533.67	6.21	7804.48
C2000.5	12.06	254.64	69.84	1615.51	235.77	2502.42	138.20	7363.48
C2000.9	25.61	758.2	293.24	487.32	26.42	1120.01	9227.71	10618.2
dsjc500_5	27.14	121.22	11.75	34.89	26.2	152.93	129.96	9248
dsjc1000_5	13.7	118.68	6.99	458.99	163.15	588.34	251.10	7740.44
MANN_a27	9.86	134.05	36.05	186.26	12.57	196.46	3359.49	59690.7
MANN_a45	9.34	170.9	84.72	481.46	15.55	557.03	161.60	37793.6

the exact algorithm, for most of the considered graphs, in terms of the quality of the solution obtained within the allowed time limit. For the larger and denser instances (second set), the exact algorithm was not able to find an optimal solution within the reported time period for all instances. Moreover, the solution it was able to detect within that time limit is still worse than the best found and even average solution obtained by heuristics for most of the instances.

Among the heuristics, the overall performance of TABU-risk and STABULUS-risk was much better than that of GRASP-risk. For the second set of graphs, GRASP-risk yielded worse average solutions for all of the instances, and worse best found solutions

for most of the instances. For the first set, however, GRASP-risk yielded solutions of quality comparable to those obtained using either TABU-risk or STABULUS-risk. In terms of the total running time, GRASP-risk was slower than the other two approaches for nearly half of all the instances tested.

As for TABU-risk and STABULUS-risk comparison, we can conclude that TABU-risk tends to yield better solutions for dense graphs as opposed to the sparse ones. It can be explained by the fact that there are no restarts in TABU-risk and, hence, it cannot explore dense regions of the sparse graph located far from the initial seed. It can be seen from Tables 2.1 and 2.2 that TABU-risk was still able to find the optimal solution in the sparse graphs in some of the runs. Therefore, with a slight modification including random restarts it can be effectively used for sparse graphs. The running time of TABU-risk was comparatively similar to that of STABULUS-risk: slower for approximately 30% of the instances, faster for 45%, and roughly the same for the rest.

Additionally, in Tables 2.5 and 2.6 we compare results of experiments performed using TABU-risk and Exact algorithm with sample size for CVaR estimation $|S| = 100m_C^2$ and $|S| = 10m_C^2$. The first column of both tables contains the name of the instance. Table 2.5 reports the robust clique sizes detected. The next two sets of two columns in this table contain the sizes of the robust cliques detected by TABU-risk for each sample size. Specifically, the average ($\overline{\omega_r}$) and the best (ω_r^{max}) solutions obtained in 50 runs are reported. Last two columns of Table 2.5 contain the size of the robust clique (ω_r) obtained by Exact algorithm for each sample size. The running time of the algorithms is reported in Table 2.6. Columns 2 through 5 of the table report the average (over the 50 runs) running time of TABU-risk for each sample size: the average time it took to reach the best found solution for the first time during each run ($\overline{t^*}$) and the average total time of algorithm execution ($\overline{T}$).

Last four columns contain the time it took to reach the best found solution for the first time (t^*) and the total running time (T) of Exact algorithm for each sample size.

Table 2.5: Comparison of robust clique sizes obtained by TABU-risk (50 runs) and the exact algorithm using sample size $|S| = 100m_C^2$ versus $|S| = 10m_C^2$ ($CVaR_{0.9}[L_C] \leq 3$ and the corresponding bounds $\omega_{p_{max}} = 4, \omega_{p_{min}} = 10$)

	TABU				Exact	
	$100m_C^2$		$10m_C^2$		$100m_C^2$	$10m_C^2$
	$\overline{\omega_r}$	ω_r^{max}	$\overline{\omega_r}$	ω_r^{max}	ω_r	ω_r
polbooks	5.62	6	5.62	6	6*	6*
football	5.84	6	5.84	6	6*	6*
celeg._metab.	6	6	6	6	6*	6*
email	6.02	7	6.02	7	7*	7*
power	4.04	6	4.04	6	6*	6*
brock200_4	7.26	8	7.12	8	7	7
brock400_2	8	8	8	8	7	7
brock400_4	8	8	8	8	7	7
brock800_2	8	8	8	8	7	7
brock800_4	8	8	8	8	7	7
C125.9	7.44	8	7.36	8	7	7
C250.9	7.98	8	7.98	8	7	7
C1000.9	8	8	8	8	7	7
C2000.5	7.96	8	7.96	8	7	7
C2000.9	8.22	9	8.16	9	7	7
dsjc500_5	7	7	7	7	7	7

From Table 2.5 we can see that the difference in the solution size obtained by TABU-risk with CVaR estimation using $|S| = 100m_C^2$ as opposed to $|S| = 10m_C^2$ was only observed for three (out of 17) instances. For Exact algorithm, no solution difference for different sample sizes has been observed. With not much difference in the accuracy of the solution obtained by using a larger sample size for CVaR estimation, we do, however, observe a drastic increase in the computational time of

TABU-risk, as seen in Table 2.6. We also observe no change in the computational time of Exact algorithm, which can be explained by the fact that it is conducted with a time limit, which is set to the same value for each experiment with $|S| = 10m^2$ and $|S| = 100m^2$. Based on the observations, derived from Tables 2.5 and 2.6, we can safely assume that $|S| = 10m^2$ is sufficient sample size to obtain accurate enough solutions within the reasonable amount of time.

Table 2.6: Comparison of the running time of TABU-risk (50 runs) and the exact algorithm using sample size $|S| = 10m_C^2$ versus $|S| = 100m_C^2$ ($CVaR_{0.9}[L_C] \leq 3$ and the corresponding bounds $\omega_{p_{max}} = 4, \omega_{p_{min}} = 10$)

	TABU				Exact			
	$100m_C^2$		$10m_C^2$		$100m_C^2$		$10m_C^2$	
	$\overline{t^*}$	$\overline{T}$	$\overline{t^*}$	$\overline{T}$	t^*	T	t^*	T
polbooks	0.01	0.01	<0.01	0.01	0.03	0.08	0.03	0.04
football	0.01	5.29	<0.01	5.34	0.01	38.27	0.01	39.89
celeg._metab.	0.01	28.17	<0.01	28.29	0.01	105.23	0.01	105.17
email	0.14	6.19	0.14	6.2	114.77	289.10	82.01	205.59
power	< 0.01	0.03	< 0.01	0.03	0.36	0.39	0.36	0.39
brock200_4	79.43	1806.98	4.71	62.27	192.84	7246.67	192.25	8473.67
brock400_2	43.45	5119	7.99	152.22	45.47	7505.46	45.40	7540.08
brock400_4	76.26	5167.62	10.76	154.88	184.83	7217.87	183.80	7255.16
brock800_2	70.26	5179.67	6.72	153.32	31.67	7535.02	31.64	7375.88
brock800_4	79.11	5190.89	9.91	157.51	313.62	8492.52	313.98	8578.13
C125.9	86.54	2597.2	10.73	95.68	53.81	9733.4	52.10	9548.19
C250.9	51.75	4872.09	11.21	176.33	115.77	7513.95	115.19	7438.79
C1000.9	14.77	2457.94	2.65	161.52	367.43	9385.47	368.94	9278.53
C2000.5	167.56	4745.25	14.47	154.58	56.01	8223.86	56.19	8024.46
C2000.9	126.19	3114.42	16.95	171.02	26.33	8334.63	26.05	7515.24
dsjc500_5	2.27	715.62	2.29	32.28	0.87	7267.01	0.81	7208.39

3. BICONNECTED AND FRAGILE SUBGRAPHS OF LOW DIAMETER

In this chapter, we focus on a diameter-based relaxation of a clique, an s-club. An s-club is a subset of vertices inducing a subgraph with a diameter of at most s. We focus on its special case with $s = 2$, study its connectivity properties, and introduce two additional structures, a *biconnected* 2-club and its opposite, a *fragile* (not biconnected) 2-club. Furthermore, we propose a combinatorial branch-and-bound algorithm to find a *maximum biconnected 2-club*, and design a polynomial time algorithm for finding a *maximum fragile 2-club* in a given graph. In addition, we formulate the maximum biconnected 2-club problem as a linear 0-1 program and solve this formulation by a branch-and-cut approach where some nontrivial constraints are applied in a lazy fashion. Finally, numerical results obtained using the proposed algorithms on a test-bed of randomly generated instances and real-life graphs are also provided.

3.1 Introduction

The concept of an s-club is commonly used to characterize network clusters in applications for which easy reachability between group members is of high importance. Even given such an advantageous property, s-clubs cannot always guarantee high connectivity and robustness. For example, an extreme case of a 2-club – a *star*, which is a graph with one "hub" vertex connected to the rest of the vertices that are not directly connected to each other, is apt to complete disconnection due to the failure of the hub [62]. Therefore, in the situations where failures are anticipated, it is important to design or detect clusters not only of small diameter, but also with additional constraints on their connectivity. To address this concern, several approaches based on the notions of vertex connectivity, heredity, and robustness can

be used.

Recall that the *(vertex) connectivity* $\kappa(G)$ of a graph G is determined by the smallest number of vertices that need to be removed from the graph to disconnect it. It also corresponds to the smallest number of vertex-disjoint paths between each pair of vertices. A graph G is said to be *k-connected* if $\kappa(G) \geq k$. An s-club S is called

- a *k-connected s-club* if $\kappa(G[S]) \geq k$.
- a *k-hereditary s-club* [62] if $diam(G[S \setminus S']) \leq s$ for any $S' \subset S$ such that $|S'| \leq k$.
- an *R-robust s-club* [71] if there are at least R vertex-disjoint paths of length at most s between all pairs of vertices in $G[S]$.

To highlight the differences between the three models, consider the complete bipartite graph $K_{3,3}$. The set of vertices of this graph is a 3-connected 2-club since we need to remove at least 3 vertices to disconnect the residual graph. It is also a 2-hereditary 2-club (we can remove up to two vertices and still preserve the diameter of two) and a 1-robust 2-club (there is only one path of length no more than two between pairs of vertices in different parts). Hence, k-connectivity is the least restrictive of the three requirements used to enforce connectivity of an s-club.

To the best of our knowledge, only one of the three models above, the R-robust s-club, has been studied from an optimization perspective [71]. In this work, we start investigating the k-connected s-club model by considering a special case of $k = s = 2$, that is, *biconnected* 2-clubs. In addition, we study the 2-clubs that are not biconnected, which we call *fragile*. Next, we formally define these two structures.

Definition 4. *Given a simple graph $G = (V, E)$, a set $S \subseteq V$ is a biconnected 2-club if $diam(G[S]) \leq 2$ and $\kappa(G[S]) \geq 2$.*

Definition 5. *Given a simple graph $G = (V, E)$, a set $S \subseteq V$ is a fragile 2-club if $diam(G[S]) \leq 2$ and $\kappa(G[S]) = 1$.*

The maximum s-club problem, which is to find an s-club of the largest cardinality in the graph, is NP-hard for any fixed s [12], even when restricted to graphs of diameter $s + 1$ [8]. We show that a maximum fragile 2-club can be detected in polynomial time, while finding a maximum biconnected 2-club is still hard.

The maximum s-club problem has been approached using heuristics [11, 67] and exact algorithms [12, 52, 38]; see [68] for a recent survey. [11] proposed three constructive heuristic methodologies for the maximum s-club problem, CONSTELLATION, DROP, and s-CLIQUE & DROP. The CONSTELLATION heuristic is based on the fact that the star graph is a 2-club. The initial solution is the largest star, which is then iteratively merged with $s-2$ more neighboring stars to result in an s-club. The DROP heuristic starts with the whole graph and iteratively removes vertices from it until the remaining set is an s-club. Finally, in the s-CLIQUE & DROP technique, the DROP heuristic is used to find an s-club in the subgraph induced by the largest s-clique, detected beforehand (a set $S \subseteq V$ is called an s-clique in G if $d_G(u, v) \leq s$ for all vertices u and v in S). CONSTELLATION and DROP heuristics have also been utilized as a lower bound estimator within the exact branch-and-bound frameworks in [12, 52]. A heuristic based on the variable neighborhood search [55] was developed by [67]. The authors used it as a lower bound estimator within the combinatorial branch-and-bound algorithm developed by [52], as well as part of a hybrid algorithm.

The maximum biconnected 2-club problem (i.e., to find a largest biconnected

2-club in a graph) is NP-hard, as discussed later in this article. However, we will show that checking biconnectivity of a given 2-club can be done in linear time. Hence, the combinatorial branch-and-bound framework for finding a maximum s-club developed by [52] can be easily adapted to detect a maximum biconnected 2-club. Intractability of the maximum biconnected 2-club problem further motivates developing integer programming techniques to solve this problem. To this aim, we will formulate this problem as a linear 0-1 program and solve this formulation by a lazy-fashioned branch-and-cut approach. We will also show that the maximum fragile 2-club problem is polynomial-time solvable and design the corresponding solution procedure.

The remainder of this chapter is organized as follows. Section 3.2 discusses the computational complexity of the maximum biconnected 2-club problem and presents some structural properties of biconnected and fragile 2-clubs. These properties are used in developing algorithms for the maximum fragile 2-club and the maximum biconnected 2-club problems in Sections 3.3 and 3.4, respectively. In particular, a polynomial-time algorithm for finding a maximum fragile 2-club in a graph is given in Section 3.3. Furthermore, a combinatorial branch-and-bound algorithm and a lazy-fashioned branch-and-cut approach based on a linear 0-1 programming formulation for the maximum biconnected 2-club problem are presented in Section 3.4. Results of the computational experiments testing the performance of the proposed algorithms on a testbed of randomly generated and real-life instances are reported in Section 4.4.

3.2 Computational complexity and structural properties

In this section, we address the computational complexity of the maximum biconnected 2-club problem and show that the decision version of this problem is NP-complete. We also further study the biconnectivity property in 2-clubs and present

some characterizations of this property. The theoretical results developed here will be used to propose exact algorithms for finding a largest biconnected and a largest fragile 2-club in a graph.

3.2.1 Computational complexity of detecting a maximum biconnected 2-club

The maximum biconnected 2-club problem is to find a biconnected 2-club of the largest size in the graph. The decision version of the maximum biconnected 2-club is given by a graph $G = (V, E)$, a positive integer c, and a question that asks if there exists a biconnected 2-club of size at least c in graph G. The following theorem addresses the computational complexity of this decision problem and shows its intractability.

Proposition 2. *The decision version of the maximum biconnected 2-club problem is NP-complete.*

PROOF. The same construction as used by [8] to show the NP-completeness of the maximum s-club problem for graphs of diameter at least s can be utilized to prove this statement. According to [8] and assuming $s = 2$, given an instance of the maximum clique problem $< G, k >$, we construct $G' = (V', E')$, such that $V' = V \cup E$ and $E' = E_1 \cup E_2$, where $E_1 = \{(v, e) : v \in V, e \in E, v \text{ is incident to } e\}$, $E_2 = \{(e_1, e_2) : e_1, e_2 \in E, e_1 \neq e_2\}$. It can be shown that G has a clique of size at least k if and only if G' has a biconnected 2-club of size at least $k + |E|$. □

3.2.2 Characterizing biconnectivity in 2-clubs

According to Menger's Theorem [53, 21], the local connectivity between two vertices is determined by finding the number of vertex-disjoint paths between them. Then, to determine the vertex connectivity of the whole graph, one needs to examine all pairs of non-adjacent vertices and find their corresponding local connectivity.

The pair with the smallest local connectivity will yield the graph connectivity number. Such procedure runs in polynomial time.

Naturally, to check if a graph is k-connected, one needs to compute its connectivity number and compare it to k. However, biconnectivity of the subgraph induced by a 2-club can be verified much faster and easier according to the following observation.

Given a set $S \subseteq V$, let us refer to a vertex in S that is adjacent to all other vertices in S as a *hub* vertex in S. A necessary and sufficient condition for a set $S \subseteq V$ to form a fragile 2-club is presented in Proposition 3.

Proposition 3. *A set $S \subseteq V$ is a fragile 2-club if and only if there exists a unique hub vertex in $G[S]$, which is also a unique cut vertex of this graph.*

PROOF. Suppose $S \subseteq V$ is a fragile 2-club. Then there is a cut vertex c in $G[S]$. Assume that it is not a hub vertex. Then, there exists a vertex $v \in S$ that is not adjacent to c. Let C_1, C_2, ..., C_p denote the set of all connected components in $G[S \setminus \{c\}]$. Note that since c is a cut vertex in $G[S]$, then $p \geq 2$. Let C_q denote the connected component that contains v. Since $p \geq 2$, then there exists a vertex $u \in C_r$, where $r \neq q$. Now, any path between u and v in $G[S]$ passes through vertex c, and since c is not adjacent to v, then the length of a shortest path between u and v in $G[S]$ will be at least three, contradicting the fact that S is a 2-club. Therefore, any cut vertex is also a hub vertex. Also, note that any graph with more than one hub vertices is 2-connected. This implies the uniqueness of the cut/hub vertex in $G[S]$, which establishes the necessity.

The proof of sufficiency is trivial and follows from the definition of a fragile 2-club and the fact that presence of a hub in a subgraph implies that its set of vertices is a 2-club. □

Corollary 1. *Given a graph $G = (V, E)$, let S be a set of vertices forming a 2-club. Then $G[S]$ is biconnected if and only if one of the following conditions holds:*

(1) there are no hub vertices in S;

(2) there are more than one hub vertex in S;

(3) there is one hub vertex c in S, but $G[S \setminus \{c\}]$ is connected..

PROOF. Follows from Proposition 3 by observing that any 2-club is either fragile or biconnected. □

Therefore, to verify whether a given 2-club is biconnected, one can check in linear time if there is a unique hub vertex in it. If the answer is "no", one can conclude that the given 2-club is biconnected. If, however, there is a unique central vertex, one can apply a breadth-first search procedure, which runs in linear time, to check whether removal of this vertex disconnects the remaining vertices. Hence, the biconnectivity of a 2-club can be verified in linear time.

Proposition 3 serves as a basis for a polynomial-time algorithm for the maximum fragile 2-club problem given in Section 3.3. In addition, it yields an interesting observation regarding the detection of a maximum biconnected 2-club in the graph induced by a fragile 2-club, presented in Corollary 2.

Corollary 2. *Let S be a set of vertices forming a fragile 2-club with c as its unique hub vertex and $C \subset S$ be the set of vertices of a largest connected component of $G[S \setminus \{c\}]$. Then, $C \cup \{c\}$ is a maximum biconnected 2-club in $G[S]$.*

PROOF. It is easy to verify that set $C \cup \{c\}$ forms a biconnected 2-club in $G[S]$. Also note that any maximal biconnected 2-club B in $G[S]$ must contain c, and $G[B \setminus \{c\}]$ must be connected to ensure biconnectivity. Hence, a connected component of the

largest size together with c yields a biconnected 2-club of the largest possible size in $G[S]$. □

Using the results in Corollaries 1 and 2, we can now adapt the combinatorial branch-and-bound framework proposed by [52] to develop an exact algorithm for solving the maximum biconnected 2-club problem (see Section 3.4.1). In addition, Corollary 1 is used to enhance the performance of the branch-and-cut approach discussed in Section 3.4.2.

3.3 A polynomial-time algorithm for the maximum fragile 2-club problem

According to Proposition 3, a fragile 2-club will always have a unique hub vertex, which is also the only cut vertex in the corresponding induced subgraph. Therefore, the maximum size of a fragile 2-club that contains $i \in V$ as its unique hub vertex is equal to $1 + \alpha_i$, where α_i is the maximum size of a subset of $N_G(i)$ that induces a disconnected subgraph. In other words, the maximum size of a fragile 2-club in $G[N_G[i]]$ can be determined as the difference between the size of the closed neighborhood of i, $|N_G[i]|$, and the connectivity of the subgraph induced by the open neighborhood of i, $\kappa(G[N_G(i)])$. Then, by detecting a vertex i with the largest value of $|N_G[i]| - \kappa(G[N_G(i)])$, the maximum fragile 2-club problem can be solved. The outline of this simple procedure is presented in Algorithm 8. Note that the connectivity of a graph can be calculated in polynomial time (e.g., $O(|E|)$ using Ford-Fulkerson algorithm [17]). Thus, the algorithm runs in $O(|V||E|)$ time.

3.4 Algorithms for solving the maximum biconnected 2-club problem

In this section, we employ the theoretical results established above regarding the biconnectivity property in 2-clubs to solve the maximum biconnected 2-club problem using a combinatorial branch-and-bound algorithm and a branch-and-cut method employing some of the nontrivial constraints of the corresponding integer

Algorithm 8 An algorithm for finding a largest fragile 2-club

Input: $G = (V, E)$
Output: A maximum fragile 2-club $S^* \subseteq V$

$max \leftarrow 0$
for each vertex $i \in V$ **do**
  $\kappa(G[N_G(i)]) \leftarrow$ connectivity of $G[N_G(i)]$
  $f \leftarrow |N_G[i]| - \kappa(G[N_G(i)])$
  if $f > max$ **then**
    $max \leftarrow f$
    $i^* \leftarrow i$
    $VC \leftarrow$ min vertex cut in $G[N_G(i^*)]$
$S^* \leftarrow N_G[i^*] \setminus VC$
return S^*

programming formulation in a lazy manner.

3.4.1 A combinatorial branch-and-bound algorithm

To find a maximum biconnected 2-club in a graph, we use a branch-and-bound (BB) algorithm originally developed for the maximum s-club problem by [52]. The general idea of this algorithm is as follows. Two sets are maintained at each BB node: a fixed set F, and an unfixed set U. Set F contains all the vertices selected to be in the solution throughout the unique path from the root node of the BB tree to the present node using a simple variable dichotomy branching rule. Set U, referred to as the candidate list, contains vertices that are at distance at most s from all vertices in F. Note that the vertices in F do not necessarily form an s-club. However, they have to be at distance at most s from each other in the subgraph induced by the union of F and U. Otherwise, there will be no s-club in the tree rooted at the present BB node that contains all vertices in F. In other words, set F should form an s-clique in $G[F \cup U]$.

When branching, a vertex from U is either deleted or moved to F, and set U is updated. If after these operations, set F does not form an s-clique in $G[F \cup U]$, the

BB node is fathomed by *infeasibility*. Otherwise, an upper bound on the cardinality of the largest s-club in $G[F \cup U]$ is estimated. If this bound is not larger than the cardinality of the incumbent solution, then the BB node is fathomed by *bound*. When $F \cup U$ becomes an s-club, the node is fathomed by *feasibility* and the incumbent solution is updated if necessary.

The adaptation of this algorithm to the maximum biconnected 2-club problem, referred to as BB2, is outlined in Algorithm 9, and the corresponding modifications are discussed in the following three subsections.

3.4.1.1 Lower bound heuristics

To initiate the incumbent solution in BB2, the following two heuristics are used for finding a biconnected 2-club in the input graph.

VDEGREE sorts vertices in a non-increasing order of their degree, and then explores connected components of the subgraphs induced by the open neighborhood of each vertex. Throughout the heuristic execution, the largest connected component found is recorded. When its size becomes larger than the degree of the next vertex to be evaluated, the heuristic terminates. The largest connected component detected along with its corresponding vertex forms a biconnected 2-club.

The other technique is the DROP heuristic, originally proposed for s-club detection by [11] and later successfully used as part of the lower bound heuristics for BB algorithms in [12, 52, 67]. It starts with the initial graph and iteratively removes vertices with the smallest number of s-neighbors (vertices located at distance at most s from a vertex) until an s-club is detected. Previous experiments in the literature [11] and our own preliminary results showed that this heuristic tends to perform better than the others when used on instances with higher edge density. However, the output of the DROP heuristic is not always a biconnected 2-club. If

Algorithm 9 BB2: A combinatorial branch-and-bound algorithm for the maximum biconnected 2-club problem

Input: $G = (V, E)$
Output: A maximum biconnected 2-club $S^* \subseteq V$

$S^* \leftarrow$ initial heuristic solution (Section 4.2.4); $LB \leftarrow |S^*|$
$Tree \leftarrow \emptyset$ ▷ Initialize the BB tree
$U_x \leftarrow V, F_x \leftarrow \emptyset$ ▷ Create the first node x of the BB tree
$Tree \leftarrow Tree \cup \{x\}$
while $Tree \neq \emptyset$ **do**
  Pick a node $x \in Tree$
  $Tree \leftarrow Tree \setminus \{x\}$
  PROCESS(x)
  if F_x is not a 2-clique in $G[F_x \cup U_x]$ **then** fathom x by infeasibility
  else
    $UB_x \leftarrow$ upper bound (Section 4.2.3)
    if $UB_x \leq LB$ **then** fathom x by bound
    else
      if $F_x \cup U_x$ is a 2-club **then**
        if $F_x \cup U_x$ is biconnected (Section 3.4.1.3) **then**
          $S^* \leftarrow F_x \cup U_x$; $LB \leftarrow |S^*|$
        else ▷ $F_x \cup U_x$ is a fragile 2-club and c is its unique hub vertex
          $C \leftarrow$ largest connected component in $G[(F_x \cup U_x) \setminus \{c\}]$ (Corollary 2)
          if $|C| + 1 > LB$ **then**
            $S^* \leftarrow C \cup \{c\}$; $LB \leftarrow |S^*|$
        Fathom x by feasibility
      else
        BRANCH($x, Tree$)
return S^*

procedure PROCESS(x)
  $U_{DROP} \leftarrow \{i \in U_x :$ there exists $j \in F_x$ such that $d_{G[F_x \cup U_x]}(i, j) > 2\}$
  while $U_{DROP} \neq \emptyset$ **do**
    $U_x \leftarrow U_x \setminus U_{DROP}$
    $U_{DROP} \leftarrow \{i \in U_x :$ there exists $j \in F_x$ such that $d_{G[F_x \cup U_x]}(i, j) > 2\}$

procedure BRANCH($x, Tree$)
  Pick vertex $u \in U_x$
  Initialize two child nodes of x, namely x' and x'':
  $U_{x'} \leftarrow U_x \setminus \{u\}, F_{x'} \leftarrow F_x \cup \{u\}$
  $U_{x''} \leftarrow U_x \setminus \{u\}, F_{x''} \leftarrow F_x$
  $Tree \leftarrow Tree \cup \{x'\} \cup \{x''\}$

this is the case, the output of VDEGREE is used as the initial incumbent solution. If the DROP heuristic does yield a feasible solution, then the largest of VDEGREE

and DROP results is selected.

3.4.1.2 Upper bounding approach

For an upper bound estimation, we used one of the techniques from [52], which is based on the fact that the *chromatic number* of the s-th power graph G^s is an upper bound on the size of the largest s-club in graph G. Given $G = (V, E)$, the s-th power graph is defined as $G^s = (V, E^s)$, where $E^s = \{(u, v) : u, v \in V; d_G(u, v) \leq s\}$. The *chromatic number* of a graph is the smallest number of *colors* required to assign to each vertex, such that no adjacent vertices have the same color. It is easy to see that this result also holds for a biconnected or fragile version of an s-club. The upper bound is computed at each BB2 tree node for the subgraph induced by set $F \cup U$ associated with this node. Finding the chromatic number of a graph is an NP-hard problem [31]. Therefore, instead of solving this problem to optimality at each node of BB2 tree to find an upper bound, a feasible coloring is obtained by a combination of a greedy heuristic and DSATUR method [13], as described in [52].

3.4.1.3 Biconnectivity property

In BB2, when a 2-club is detected, its biconnectivity is verified in linear time by using the conditions of Corollary 1. If biconnectivity is confirmed, the node is fathomed by feasibility and the incumbent solution is updated. If the detected 2-club (set $F \cup U$) is not biconnected, then using Corollary 2, a maximum biconnected 2-club is found in the graph induced by this 2-club. Here, denoting the hub vertex in $G[F \cup U]$ by c, the breadth-first search is used to detect in linear time a largest connected component in $G[(F \cup U) \setminus \{c\}]$, which united with the hub vertex c forms a largest biconnected 2-club in $G[F \cup U]$. After obtaining a largest biconnected 2-club in $G[F \cup U]$, the BB2 node is fathomed by feasibility and the incumbent solution is updated if necessary.

3.4.2 A branch-and-cut algorithm

In order to develop a branch-and-cut (BC) algorithm for solving the maximum biconnected 2-club problem, we first formulate this problem as a linear 0-1 program. To this aim, we first need to present the following definitions. Given a graph $G = (V, E)$ and a pair of non-adjacent vertices $i, j \in V$ that are connected in G, an (i, j)-*vertex-cut* in G is a subset of vertices $C \subseteq V \setminus \{i, j\}$ such that i and j are disconnected in $G[V \setminus C]$. A *minimal* (i, j)-*vertex-cut* in G is an (i, j)-vertex-cut in this graph that is not a proper superset of a smaller (i, j)-vertex-cut. Finally, a *minimum* (i, j)-*vertex-cut* in G is a minimal (i, j)-vertex-cut of the smallest cardinality in G. Recall also that the size of the minimum (i, j)-vertex-cut in G corresponds to the number of vertex-disjoint paths between i and j in G, and is also known as the connectivity between i and j in G.

Let Π_G^{ij} denote the set of all minimal (i, j)-vertex-cuts in G. Next, we define a decision vector $\mathbf{x} = (x_1, ..., x_{|V|})$ such that:

$$x_i = \begin{cases} 1, & \text{if vertex } i \text{ belongs to the biconnected 2-club,} \\ 0, & \text{otherwise.} \end{cases}$$

Then, the maximum biconnected 2-club problem can be formulated as follows:

$$\max \quad \sum_{i \in V} x_i \tag{3.1a}$$

$$\text{s.t.} \quad x_i + x_j - \sum_{k \in N_G(i) \cap N_G(j)} x_k \leq 1, \quad \forall (i,j) \notin E, \tag{3.1b}$$

$$2x_i + 2x_j - \sum_{k \in C} x_k \leq 2, \quad \forall i,j \in V \text{ with } d_G(i,j) = 2, C \in \Pi_G^{ij}, \tag{3.1c}$$

$$x_i \in \{0,1\}, \quad \forall i \in V. \tag{3.1d}$$

To show the validity of this formulation, we need to prove that constraints (3.1b)-(3.1d) correctly characterize the set of feasible solutions to the maximum biconnected 2-club problem. First, given the incidence vector of a biconnected 2-club $\hat{S}$ (denoted by $\hat{\mathbf{x}}$), we show that $\hat{\mathbf{x}}$ satisfies constraints (3.1b)-(3.1d). Note that constraints (3.1b) and (3.1d) are satisfied by $\hat{\mathbf{x}}$ since it is the incidence vector of a 2-club [8]. Now, consider a pair of vertices $i,j \in V$ such that $d_G(i,j) = 2$. If $\hat{x}_i + \hat{x}_j \leq 1$, then constraint (3.1c) is trivially valid for all $C \in \Pi_G^{ij}$. Suppose $\hat{x}_i + \hat{x}_j = 2$ (i.e., $\hat{x}_i = \hat{x}_j = 1$) and there exists a set $\hat{C} \in \Pi_G^{ij}$ for which constraint (3.1c) is violated. So, $\sum_{k \in \hat{C}} \hat{x}_k \leq 1$ and $|\hat{C} \cap \hat{S}| \leq 1$. Since $\hat{C}$ is an (i,j)-vertex-cut in G, then $\hat{C} \cap \hat{S}$ is also an (i,j)-vertex-cut in $G[\hat{S}]$. Now, having $|\hat{C} \cap \hat{S}| \leq 1$ contradicts the fact that $\hat{S}$ is a biconnected 2-club. Therefore, $\hat{\mathbf{x}}$ satisfies constraints (3.1b)-(3.1d).

Now, let $\bar{\mathbf{x}} \in \{0,1\}^{|V|}$ denote a binary vector satisfying constraints (3.1b) and (3.1c). We show that $\bar{\mathbf{x}}$ is the incidence vector of a biconnected 2-club. First, since constraints (3.1b) are satisfied by $\bar{\mathbf{x}}$, then $\bar{\mathbf{x}}$ is the incidence vector of a 2-club [8]. Assume that $\bar{\mathbf{x}}$ corresponds to a fragile 2-club $\bar{S}$. Then, by Proposition 1, there exists a unique hub vertex c in $G[\bar{S}]$, which is also the unique cut vertex of $G[\bar{S}]$. So, there exists a pair of non-adjacent vertices $u,v \in \bar{S} \setminus \{c\}$ that are disconnected in $G[\bar{S} \setminus \{c\}]$.

We know that $(V \setminus \bar{S}) \cup \{c\}$ is a (u, v)-vertex-cut in G. Let $\bar{C} \subseteq (V \setminus \bar{S}) \cup \{c\}$ be a minimal (u, v)-vertex-cut in G. Note that $\bar{C} \cap \bar{S} = \{c\}$. This means $\sum_{k \in \bar{C}} \bar{x}_k = 1$ and constraint (3.1c) corresponding to vertices $u, v \in V$ and $\bar{C} \in \Pi_G^{uv}$ is violated by $\bar{\mathbf{x}}$, which is a contradiction.

Even though Formulation (4.1) may seem impractical due to the challenge associated with the enumeration of all minimal vertex cuts between all pairs of vertices with distance 2 and the potentially very large number of the corresponding constraints, its implementation can be rather simple by employing a lazy-fashioned branch-and-cut algorithm. To this aim, we relax Formulation (4.1) by removing constraints (3.1c) and only add some of them (implemented as lazy constraints added via a callback function in Gurobi Optimizer [37]) whenever an incumbent solution is a fragile 2-club. Recall that (4.1) is just a formulation for the maximum 2-club problem if constraints (3.1c) are not included [8]. We can also take advantage of the results of Corollary 1 to verify the biconnectivity of an incumbent 2-club. In case an incumbent 2-club is fragile, the procedure described in Algorithm 10 is used to detect a violated constraint of type (3.1c). This violated constraint is then added to the problem formulation at the current search-tree node to cut off the infeasible incumbent solution and further process this node of the search-tree.

Note that Algorithm 10 runs in $O(|V|^2)$, which can be verified as follows. Its bottleneck is the construction of a minimal (u, v)-vertex-cut. At each iteration of the for-loop in Algorithm 10, a breadth-first search is used to verify if u and v are disconnected in $G[W \cup \{i\}]$, which runs in linear time and is repeated for at most $O(|V|)$ iterations. Hence, the complexity of a minimal (u, v)-vertex-cut construction is $O(|V|^2)$. Also, it is easy to see that Algorithm 10 yields a correct violated constraint of type (3.1c) for the incidence vector of S (denoted by $\tilde{x}$). Notice that $\tilde{x}_u = \tilde{x}_v = 1$,

Algorithm 10 A procedure for detecting a violated constraint of type (3.1c) for a given fragile 2-club

Input: $G = (V, E)$; a fragile 2-club $S \subseteq V$
Output: $u, v \in V$ such that $d_G(u, v) = 2$ and a set $\bar{C} \in \Pi_G^{uv}$ associated with a violated constraint of type (3.1c) for the incidence vector of S

$c \leftarrow$ hub vertex in S
Select any pair of vertices $u, v \in S \setminus \{c\}$ that are disconnected in $G[S \setminus \{c\}]$
$C \leftarrow (V \setminus S) \cup \{c\}$ ▷ Note that C is a (u, v)-vertex-cut.
$\bar{C} \leftarrow \emptyset$ ▷ Finding a minimal (u, v)-vertex-cut $\bar{C} \subset C$ starts.
$W \leftarrow S \setminus \{c\}$
for each vertex $i \in C$ **do**
  if u and v are disconnected in $G[W \cup \{i\}]$ **then**
    $W \leftarrow W \cup \{i\}$
  else
    $\bar{C} \leftarrow \bar{C} \cup \{i\}$
return u, v, and $\bar{C}$ ▷ Finding a minimal (u, v)-vertex-cut $\bar{C} \subset C$ terminates.

$\bar{C} \cap S = \{c\}$, $\sum_{k \in \bar{C}} \tilde{x}_k = 1$ and constraint

$$2x_u + 2x_v - \sum_{k \in \bar{C}} x_k \leq 2$$

is violated by $\tilde{x}$.

3.5 Computational experiments

To test the performance of the proposed algorithms, a set of randomly generated graph instances from [52, 67] and selected real-life graphs from 10$^{\text{th}}$ DIMACS Implementation Challenge [23] were used. All algorithms were coded in C++ and compiled using Microsoft Visual Studio 2010. The lazy-fashioned BC algorithm was implemented using Gurobi Optimizer 6.5 [37]. All experiments were performed on a PC with 2.40 GHz CPU and 12 GB RAM, running Windows 7 Enterprise.

The random instances had been generated according to the approach introduced by [33] using two density parameters a and b ($0 \leq a \leq b \leq 1$). This method yields random graphs with the *expected edge density* $D = (a + b)/2$, and the *vertex degree*

variance equal to $b - a$, which reaches its minimum value when $a = b = D$, and maximum – when $a = 0, b = 2D$. The instances used range in size (50, 100, 150, and 200 vertices), edge density (0.0125, 0.025, 0.5, 0.1, 0.15, 0.2, and 0.25) and vertex degree variance (minimum and maximum). There are ten graph instances for each combination of the aforementioned parameters, and only the average results obtained across each set of the ten instances are reported.

Tables 3.1 and 3.2 report the results of the experiments performed on the above-mentioned set of graphs for the minimum and maximum vertex degree variance, respectively. In each table, the first two columns contain the general information about the instance (the size of the vertex set and the edge density). The next set of three columns reports information regarding the performance of the BB algorithm for finding a maximum 2-club. Namely, the first column contains the average (over the ten instances) size of the largest 2-clubs obtained, the average optimality gap, and the average running time of the BB algorithm proposed by [52]. The optimality gap is computed as a ratio of the difference between an upper and a lower bound to the upper bound ($(UB - LB)/UB$), converted to a percentage. If for a given set of instances the average optimality gap equals zero, it means that the optimal solution to the maximum 2-club problem was detected for each of the ten instances. The next set of three columns contains the same information (the average size, gap and the running time) for the biconnected 2-club problem solved using the BB2 algorithm. The next two columns show the average size of the largest fragile 2-clubs found by Algorithm 8 and the corresponding running time. The last column contains the average size of the 2-clubs formed by the vertices from the closed neighborhood of the largest degree (Δ) vertex in the graph.

Note that some entries in Table 3.2 (also Tables 3.4-3.5) report the optimality gap larger than zero, but the run time smaller than the limit of 3600 seconds. This

Table 3.1: Results of the experiments on a set of random instances with the minimum vertex degree variance.

$\|V\|$	Density	**2-club (BB)**			**biconnected 2-club (BB2)**			**fragile 2-club (Alg. 8)**		$\Delta + 1$
		Size	Gap(%)	CPU(s)	Size	Gap(%)	CPU(s)	Size	CPU(s)	
50	0.0125	3.7	0	0.01	2.0	0	0.22	3.7	<0.01	3.7
	0.025	4.9	0	0.07	2.9	0	1.37	4.9	<0.01	4.9
	0.05	6.7	0	0.64	5.6	0	2.37	6.7	<0.01	6.7
	0.1	11.0	0	0.58	9.8	0	1.63	11.0	<0.01	11.0
	0.15	15.5	0	0.90	15.2	0	0.65	14.9	0.01	15.0
	0.2	23.5	0	1.69	23.5	0	5.07	16.9	0.06	17.5
	0.25	34.6	0	5.63	34.6	0	6.41	19.1	0.16	20.1
100	0.0125	6.1	0	0.02	2.6	0	2.30	6.1	<0.01	6.1
	0.025	8.8	0	0.08	5.6	0	18.60	8.8	<0.01	8.8
	0.05	11.6	0	13.19	9.2	0	14.86	11.6	<0.01	11.6
	0.1	18.9	0	32.50	16.1	0	51.89	18.9	<0.01	18.9
	0.15	29.8	0	1327.33	29.8	0	1321.10	25.4	0.11	25.9
	0.2	71.2	0	723.97	71.2	0	640.35	29.0	0.92	30.9
	0.25	95.9	0	3.36	95.9	0	4.10	32.9	2.78	36.9
150	0.0125	6.9	0	0.05	5.2	0	18.11	6.9	<0.01	6.9
	0.025	11.0	0	1.93	6.8	0	55.23	11.0	<0.01	11.0
	0.05	16.8	0	65.21	10.7	0	48.88	16.8	<0.01	16.8
	0.1	26.8	0	411.79	25.2	0	448.37	26.8	0.01	26.8
	0.15	37.7	48.4	3604.46	37.6	48.5	3604.98	34.9	0.62	35.9
	0.2	136.8	0	406.24	136.8	0	389.62	41.9	5.54	45.7
	0.25	149.4	0	2.43	149.4	0	0.85	46.6	27.77	52.0
200	0.0125	8.2	0	0.11	5.4	0	43.57	8.2	<0.01	8.2
	0.025	12.7	0	45.74	7.7	0	70.17	12.7	<0.01	12.7
	0.05	20.7	0	171.84	13.3	0	176.20	20.7	0.01	20.7
	0.1	33.6	35.8	3604.17	33.1	36.6	3603.11	33.2	0.18	33.6
	0.15	58.5	55.7	3615.15	58.5	55.7	3615.00	42.5	2.56	44.7
	0.2	195.5	0	9.04	195.5	0	12.97	53.5	11.21	57.5
	0.25	200	0	<0.01	200	0	<0.01	59.7	174.16	68.8

happens when some of the ten tested instances are solved to optimality, and some are not, but only the average values are reported.

The running time limit for both BB and BB2 algorithms, similarly to [52], was set to 3600 seconds. The limit is enforced by monitoring the elapsed time after processing

Table 3.2: Results of the experiments on a set of random instances with the maximum vertex degree variance.

$\|V\|$	Density	**2-club (BB)**			**biconnected 2-club (BB2)**			**fragile 2-club (Alg. 8)**		$\Delta+1$
		Size	Gap(%)	CPU(s)	Size	Gap(%)	CPU(s)	Size	CPU(s)	
50	0.0125	4.3	0	0.01	2.4	0	0.25	4.3	<0.01	4.3
	0.025	5.2	0	<0.01	4.8	0	0.59	4.6	<0.01	4.6
	0.05	7.4	0	0.05	5.8	0	2.82	7.4	<0.01	7.4
	0.1	11.6	0	0.57	10.4	0	2.43	11.6	<0.01	11.6
	0.15	16.3	0	0.85	16.1	0	0.54	14.7	0.02	14.9
	0.2	24.1	0	1.17	24.1	0	4.92	17.8	0.06	18.2
	0.25	34.6	0	1.27	34.6	0	1.14	21.8	0.27	23.1
100	0.0125	5.9	0	0.02	2.9	0	8.32	5.9	<0.01	5.9
	0.025	10.1	0	0.06	6.6	0	23.04	10.1	<0.01	10.1
	0.05	12.7	0	16.89	9.3	0	16.66	12.7	<0.01	12.7
	0.1	22.1	0	41.43	20.8	0	45.01	21.9	0.02	22.0
	0.15	40.7	0	540.60	40.7	0	517.33	29.6	0.11	29.9
	0.2	72.5	0	57.58	72.5	0	78.38	36.2	1.30	37.9
	0.25	88.2	0	8.17	88.2	0	11.40	40.5	5.61	44.6
150	0.0125	8.3	0	0.04	4.8	0	26.61	8.3	<0.01	8.3
	0.025	11.8	0	3.30	7.7	0	50.10	11.8	<0.01	11.8
	0.05	19.5	0	60.85	12.7	0	56.27	19.5	<0.01	19.5
	0.1	31.7	0	498.01	30.4	0	592.81	31.6	0.05	31.7
	0.15	76.1	9.0	3062.24	76.0	9.2	3064.04	41.7	1.43	43.0
	0.2	123.6	0	75.69	123.6	0	109.02	51.7	5.19	54.1
	0.25	143.8	0	3.03	143.8	0	5.04	61.3	33.20	67.2
200	0.0125	9.2	0	0.10	5.6	0	56.75	9.2	<0.01	9.2
	0.025	14.3	0	32.04	8.5	0	77.94	14.3	<0.01	14.3
	0.05	23.4	0	200.85	15.8	0	184.65	23.4	0.01	23.4
	0.1	39.1	24.9	3522.09	38.4	25.7	3582.95	38.7	0.31	39.1
	0.15	126.2	7.5	3256.49	126.2	7.5	3263.84	54.0	6.00	56.3
	0.2	178.2	0	437.15	178.2	0	568.02	66.2	35.17	70.4
	0.25	196.4	0	7.08	196.4	0	8.66	79.0	178.29	86.1

each branch-and-bound tree node. Hence, in some cases, the reported running times exceed 3600 seconds due to the delay associated with the last processed node.

As it can be seen from the tables, the time spent on finding a maximum biconnected 2-club in denser graphs is not significantly different from the time spent on

finding a maximum 2-club in such instances as the large 2-clubs in denser graphs tend to be biconnected. Therefore, the addition of extra steps to the BB algorithm to ensure the biconnectivity of a solution has not drastically impacted the algorithm's performance. Conversely, the average time spent on finding a maximum biconnected 2-club in sparser graphs seems to be in general larger then the time spent on finding a maximum 2-club in these instances as the large 2-clubs in sparser graphs tend to be fragile. As a result, more computational effort is required by BB2 to ensure the biconnectivity of a solution. This can also be verified from the results as the maximum size of a 2-club is comparable with that of a fragile 2-club in sparse graph instances, and with the maximum size of a biconnected 2-club in denser ones. It was also observed that in sparser instances, the size of a maximum fragile 2-club is the same as the size of a 2-club formed by the closed neighborhood of a maximum degree vertex. This is not however the case for denser graphs as the closed neighborhood of a maximum degree vertex in such graphs tends to be a biconnected 2-club. It should be noted that a simple heuristic based on selecting the closed neighborhood of a maximum degree vertex was shown to be "provably best" for the maximum s-club problem [43], in the sense that it is NP-hard to improve this solution whenever it is possible.

According to Tables 3.1 and 3.2, the running time of Algorithm 8 increases as the edge density increases. This means detecting a maximum fragile 2-club in denser instances requires more computational effort as expected. Considering the maximum 2-club problem and the maximum biconnected 2-club problem, the challenging densities for each graph size tested are very similar to the ones reported in [52]. Also, similar to the observation made in [52], random graphs with minimum vertex degree variance are in general more challenging than the ones with maximum vertex degree variance. This further emphasizes the similarity between challenging instances for

these two problems on random graphs.

Table 3.3: Results of the experiments on a set of real-life instances.

Instance	$\|V\|$	$\|E\|$	**2-club (BB)**			**biconnected 2-club (BB2)**			**fragile 2-club (Alg. 8)**		$\Delta+1$
			Size	Gap(%)	CPU(s)	Size	Gap(%)	CPU(s)	Size	CPU(s)	
karate	34	78	18	0	0.02	17	0	0.31	18	0.02	18
dolphins	62	159	13	0	1.61	12	0	1.83	13	0.02	13
polbooks	105	441	28	0	0.53	28	0	0.83	25	0.20	26
adjnoun	112	425	50	0	0.62	48	0	33.63	50	0.03	50
football	115	613	16	0	21.28	16	0	23.65	13	<0.01	13
jazz	198	2742	103	0	290.23	103	0	283.69	101	0.16	101
c.-m.[1]	453	2025	238	0	26.35	222	0	3266.22	238	0.51	238
email	1133	5451	72	0	42.59	69	4.2	3645.88	72	0.09	72
polblogs	1490	16715	352	0	809.21	346	1.7	4227.05	352	1.73	352
netscience	1589	2742	35	0	1.78	25	28.6	3607.23	35	0.08	35
add20	2395	7462	124	0	106.92	124	0	103.80	123	58.66	124
data	2851	15093	18	21.7	3603.96	17	26.1	3713.19	18	0.19	18
uk	4824	6837	4	50	3604.87	4	50.0	3610.84	4	0.37	4
power	4941	6594	20	0	36.16	14	30.0	3672.27	20	0.23	20
add32	4960	9462	32	0	49.79	30	6.3	3637.03	32	0.44	32
hep-th	8361	15751	51	0	164.36	45	11.8	3777.78	51	0.53	51
whitaker3	9800	28989	9	35.7	3633.41	9	35.7	3647.20	7	4.76	9
crack	10240	30380	10	33.3	3974.68	10	33.3	3642.55	9	0.62	10
PGPg.[2]	10680	24316	206	0	639.26	196	4.9	3920.75	206	0.69	206
cs34	22499	43858	5	54.6	4163.80	5	54.6	4231.41	5	1.65	5

[1]celegans-metabolic
[2]PGPgiantcompo

In Table 3.3, we present the results of the experiments performed on the real-life graph instances from 10^{th} DIMACS Implementation Challenge [23]. The first three columns in the table contain details about the graph (its name, size of the vertex and edge sets). The rest of the columns include the same information regarding the results of the experiments as the one presented in Tables 3.1 and 3.2. It should be mentioned that these real-life instances are sparse with low edge densities.

Note that, according to Table 3.3, the running time of BB2 for some of the in-

stances is significantly higher than that of BB. For those instances, an initial solution for the maximum 2-club problem obtained by the lower bound heuristic was in fact optimal. Also, due to the sparse structure of the graphs, no upper bounds larger than the size of the initial solution were detected, so all of the BB tree nodes were fathomed by bound in the early stages of the algorithm. For the maximum biconnected 2-club problem on these sparse instances, however, the size of an optimal solution is usually (as shown by experiments in Tables 3.1 and 3.2) smaller than the one for the maximum 2-club problem. Hence, the upper bound obtained by the heuristic coloring at the beginning of the algorithm was larger than the size of a solution obtained by the lower bound heuristic, which led to the creation of many BB2 tree nodes and a drastic increase in the running time.

Also, it can be seen from Table 3.3 that the size of a maximum 2-club and the size of a maximum fragile 2-club are equal for more than half of the instances and do not differ greatly for the rest, which supports our claim that in the sparse graphs a 2-club of the largest size is most likely fragile.

In the next three tables, we report the performance of the BC algorithm compared to that of BB2. The same set of graph instances as in the previous three tables is used for the experiments. The results of the experiments on the randomly generated graphs with the minimum and maximum vertex degree variance are presented in Tables 3.4 and 3.5, respectively. The results of the experiments on the real-life instances are summarized in Table 3.6. Same as in the previous tables, we report the average size of the best detected solution, the average optimality gap (computed as $(UB - LB)/UB$), and the average running time, which is limited to 3600 seconds.

According to the results presented in Tables 3.4-3.5, BC approach performs extraordinary well on all random graph instances. We observed that BC ran faster on such instances and found better solutions than those obtained by BB2 in case of

Table 3.4: Comparison of the performance of the branch-and-bound algorithm (BB2) and the branch-and-cut algorithm (BC) on a set of random instances with the minimum vertex degree variance.

$\|V\|$	Density	**BB2**			**BC**		
		Size	Gap (%)	CPU (s)	Size	Gap (%)	CPU (s)
50	0.0125	2.0	0	0.22	2.0	0	0.02
	0.025	2.9	0	1.37	2.9	0	0.07
	0.05	5.6	0	2.37	5.6	0	0.06
	0.1	9.8	0	1.63	9.8	0	0.25
	0.15	15.2	0	0.65	15.2	0	0.27
	0.2	23.5	0	5.07	23.5	0	0.20
	0.25	34.6	0	6.41	34.6	0	0.04
100	0.0125	2.6	0	2.30	2.6	0	0.45
	0.025	5.6	0	18.60	5.6	0	0.91
	0.05	9.2	0	14.86	9.2	0	0.92
	0.1	16.1	0	51.89	16.1	0	3.39
	0.15	29.8	0	1321.10	29.8	0	5.59
	0.2	71.2	0	640.35	71.2	0	0.66
	0.25	95.9	0	4.10	95.9	0	0.05
150	0.0125	5.2	0	18.11	5.2	0	1.00
	0.025	6.8	0	55.23	6.8	0	2.42
	0.05	10.7	0	48.88	10.7	0	8.68
	0.1	25.2	0	448.37	25.2	0	67.43
	0.15	37.6	48.5	3604.98	54.6	0	1410.94
	0.2	136.8	0	389.62	136.8	0	0.13
	0.25	149.4	0	0.85	149.4	0	0.15
200	0.0125	5.4	0	43.57	5.4	0	4.77
	0.025	7.7	0	70.17	7.7	0	7.57
	0.05	13.3	0	176.20	13.3	0	32.61
	0.1	33.1	36.6	3603.11	33.1	0	1959.18
	0.15	58.5	55.7	3615.00	100.6	9.4	2776.97
	0.2	195.5	0	12.97	195.5	0	0.27
	0.25	200	0	<0.01	200	0	0.34

non-optimal termination. More specifically, for some of the hard cases of random instances, for which BB2 could not reach the optimal solution or prove the optimality of the obtained solution within the time limit, BC was able to get better solutions

and smaller optimality gaps, or even obtain the optimal solution within the same time limit. We observed the same behavior of BC on the ten smallest real-life graphs in Table 3.6. For the rest of the graphs in this table (larger instances) the performance of BC deteriorated, as this algorithm could not even find a nontrivial feasible solution and optimality gap for the last two instances. This is due to the fact that Formulation (4.1) could not even be loaded to the machine's memory resulting in a crash before start of the branch-and-cut procedure.

The results presented in Tables 3.4-3.6 justify that even though BC approach for the maximum biconnected 2-club problem might be better applicable for random graphs and moderate-sized real-life instances, the large real-life graphs can only be tackled by heuristics and combinatorial branch-and-bound algorithms such as BB2.

Table 3.5: Comparison of the performance of the branch-and-bound algorithm (BB2) and the branch-and-cut algorithm (BC) on a set of random instances with the maximum vertex degree variance.

		BB2			**BC**		
$\|V\|$	Density	Size	Gap (%)	CPU (s)	Size	Gap (%)	CPU (s)
50	0.0125	2.4	0	0.25	2.4	0	0.02
	0.025	4.8	0	0.59	4.8	0	0.03
	0.05	5.8	0	2.82	5.8	0	0.12
	0.1	10.4	0	2.43	10.4	0	0.18
	0.15	16.1	0	0.54	16.1	0	0.23
	0.2	24.1	0	4.92	24.1	0	0.06
	0.25	34.6	0	1.14	34.6	0	0.02
100	0.0125	2.9	0	8.32	2.9	0	0.43
	0.025	6.6	0	23.04	6.6	0	0.87
	0.05	9.3	0	16.66	9.3	0	1.35
	0.1	20.8	0	45.01	20.8	0	2.42
	0.15	40.7	0	517.33	40.7	0	2.38
	0.2	72.5	0	78.38	72.5	0	0.13
	0.25	88.2	0	11.40	88.2	0	0.06
150	0.0125	4.8	0	26.61	4.8	0	1.85
	0.025	7.7	0	50.10	7.7	0	2.60
	0.05	12.7	0	56.27	12.7	0	7.24
	0.1	30.4	0	592.81	30.4	0	21.53
	0.15	76.0	9.2	3064.04	81.5	0	5.90
	0.2	123.6	0	109.02	123.6	0	0.17
	0.25	143.8	0	5.04	143.8	0	0.16
200	0.0125	5.6	0	56.75	5.6	0	5.12
	0.025	8.5	0	77.94	8.5	0	7.39
	0.05	15.8	0	184.65	15.8	0	30.38
	0.1	38.4	25.7	3582.95	38.9	6.9	2010.28
	0.15	126.2	7.5	3263.84	133.1	0	5.56
	0.2	178.2	0	568.02	178.2	0	0.31
	0.25	196.4	0	8.66	196.4	0	0.36

Table 3.6: Comparison of the performance of the branch-and-bound algorithm (BB2) and the branch-and-cut algorithm (BC) on a set of real-life instances.

Instance	$\|V\|$	$\|E\|$	**BB2**			**BC**		
			Size	Gap (%)	CPU (s)	Size	Gap (%)	CPU (s)
karate	34	78	17	0	0.31	17	0	0.02
dolphins	62	159	12	0	1.83	12	0	0.09
polbooks	105	441	28	0	0.83	28	0	0.10
adjnoun	112	425	48	0	33.63	48	0	0.22
football	115	613	16	0	23.65	16	0	3.57
jazz	198	2742	103	0	283.69	103	0	0.47
celegans-metabolic	453	2025	222	0	3266.22	222	0	21.25
email	1133	5451	69	4.2	3645.88	69	0	88.56
polblogs	1490	16715	346	1.7	4227.05	346	0	213.91
netscience	1589	2742	25	28.6	3607.23	25	0	127.12
add20	2395	7462	124	0	103.80	124	0	159.47
data	2851	15093	17	26.1	3713.19	15	28.6	3606.68
uk	4824	6837	4	50.0	3610.84	4	55.6	3602.06
power	4941	6594	14	30.0	3672.27	2	90.0	3609.07
add32	4960	9462	30	6.3	3637.03	13	69.0	3639.78
hep-th	8361	15751	45	11.8	3777.78	2	100.0	3602.84
whitaker3	9800	28989	9	35.7	3647.20	6	100.0	3622.76
crack	10240	30380	10	33.3	3642.55	6	100.0	3690.59
PGPgiantcompo	10680	24316	196	4.9	3920.75	-	-	-
cs34	22499	43858	5	54.6	4231.41	-	-	-

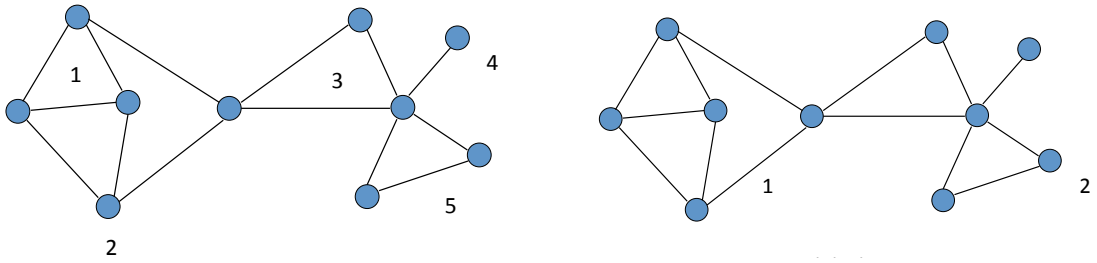
4. PARTITIONING THE GRAPH INTO S-CLUBS

In this chapter, we once again consider s-clubs, however, our focus shifts from the problem of detecting the largest s-clubs into the problem of partitioning the graph into the minimum number of non-overlapping s-clubs, referred to as the *minimum s-club partitioning* problem. Integer programming techniques and combinatorial branch-and-bound framework are employed to develop exact algorithms to solve this problem. We also study and compare the computational performance of the proposed algorithms for the special case of $s = 2$ on a test-bed of randomly generated instances and real-life graphs.

4.1 Introduction

Partitioning a graph into s-clubs is a good alternative to clique partitioning due to restrictive nature of a clique. Many naturally arising cohesive subgroups in real-life networks are not ideal and may be a few connections short of a clique structure. Moreover, if the objective of a graph partitioning is to scale down the size of the graph and obtain its hierarchical structure, s-club partitioning can yield much more meaningful results rather than clique partitioning. Consider example shown on Fig. 4.1. Notice that the clique partitioning yields a few one vertex subgraphs, while an s-club based approach partitions the graphs in a more reasonable way.

The minimum s-club partitioning problem which aims to partition the graph into the smallest number of non-overlapping s-clubs is NP-hard [20], even when restricted to bipartite and chordal graphs [1, 16]. All the solution approaches for this problem proposed in the literature are limited to special classes of graphs. Parley et al. [61] studied this problem on trees and designed a linear-time algorithm for partitioning the tree into the minimum number of subtrees each inducing a subgraph with a



(a) Clique clustering (b) 2-club clustering

Figure 4.1: Example of clique vs. 2-club partitioning.

diameter at most k. In [20], an approximation algorithm for the minimum s-club partitioning problem (referred to as *k-clustering* in the paper) with the worst-case performance ratio of at most 3 for a class of graphs with a dominating diametral path is proposed. Abbas and Stewart [1] studied the s-club partitioning problem (referred to as *partitioning vertices to minimize maximum diameter* in the paper) on interval and bipartite permutation graphs and presented linear-time solution techniques. In [28], an s-club partitioning approach is used for routing in wireless ad hoc networks and an approximation algorithm with the worst-case performance ratio of $O(s)$ for unit-disk graphs is designed.

In this chapter, we propose a general framework for solving the minimum s-club partitioning problem on any simple undirected graph. Besides being NP-hard, finding the minimum partition into s-clubs is quite challenging due to the complexity associated with detecting large s-clubs themselves. The maximum s-club problem is NP-hard for any fixed s, even when restricted to graphs of diameter $s + 1$ [8, 12]. Moreover, it is also NP-hard to test s-club maximality for any fixed integer $s \geq 2$ [52]. We design a combinatorial branch-and-bound algorithm for the general case of the minimum s-club partitioning problem. As an alternate approach, we also formulate this problem as a mixed 0-1 linear program, which can be solved by the branch-and-cut algorithms available via commercial solvers.

The rest of this chapter is organized as follows. A combinatorial branch-and-bound algorithm for finding a minimum s-club graph partitioning is proposed in Section 4.2. A mixed 0-1 linear programming formulation for the minimum s-club partitioning problem is presented in Section 4.3. We test the performance of the developed solution techniques for the special case of $s = 2$ on a test-bed of randomly generated instances and real-life networks, and report obtained results in Section 4.4.

4.2 A combinatorial branch-and-bound algorithm

In this section, we present a combinatorial branch-and-bound algorithm (BB) for solving the minimum s-club partitioning problem. Since the main goal of BB is to construct s-clubs, it will adopt a few core techniques from the algorithm developed for the maximum s-club problem by Pajouh et al. [52]. The outline of BB is summarized in Algorithm 11, and its details are discussed below.

4.2.1 Search tree structure

Each BB tree node x maintains a partial s-club partitioning represented by a collection of fixed sets, $F_x = \{F_x^0, F_x^1, \ldots, F_x^{|F_x|-1}\}$, and a collection of unfixed sets, $U_x = \{U_x^0, U_x^1, \ldots, U_x^{|F_x|-1}\}$, with $|F_x| = |U_x|$. Each fixed set $F_x^i, i = 1, \ldots, |F_x| - 1$ contains vertices assigned to be in the i-th s-club over the distinct path from the root node of the BB tree to the current node x using a certain branching rule. Every unfixed set $U_x^i, i = 1, \ldots, |F_x| - 1$ contains vertices that are candidates to be potentially added to the corresponding F_x^i and included in the i-th s-club. Set F_x^0 is always an empty set, and set U_x^0 contains all the vertices not yet assigned to any of $F_x^i, i = 1, \ldots, |F_x| - 1$. It should be noted that each unfixed set $U_x^i, i = 1, \ldots, |F_x| - 1$ is a subset of U_x^0. Some of the sets $U_x^i, i = 1, \ldots, |F_x| - 1$ might overlap, while all sets $F_x^i, i = 0, \ldots, |F_x| - 1$ are disjoint.

Each set $U_x^i, i = 1, \ldots, |U_x| - 1$ should contain vertices that are at distance of

at most s from all vertices of the corresponding F_x^i in $G[F_x^i \cup U_x^i]$. Vertices in each $F_x^i, i = 1, \ldots, |F_x| - 1$ do not necessarily have to form an s-club, but they have to be at distance of at most s from each other in the subgraph induced by the union of F_x^i and its corresponding U_x^i. Otherwise, no s-club containing all the vertices in F_x^i could be formed, and no feasible s-club partitioning could be derived in the tree rooted at the current BB node x. In other words, each set $F_x^i, i = 1, \ldots, |F_x| - 1$ should form an s-clique in $G[F_x^i \cup U_x^i]$.

Each BB tree node x also includes a set referred to as $CandUpd_x$, which contains all sets $U_x^i, i = 1, \ldots, |F_x| - 1$ that need to be updated later when node x is processed (see Algorithm 13). At the start of the BB algorithm, root node x is formed by setting $F_x = \{F_x^0\}$, $U_x = \{U_x^0\}$, and $CandUpd_x = \emptyset$, where $F_x^0 = \emptyset$ and $U_x^0 = V$. An initial incumbent solution is also found at this stage by using a proposed heuristic for this problem (see Section 4.2.3).

4.2.2 Branching and search strategies

When branching, a vertex $u \in U_x^0$ is selected and for each $i = 0, \ldots, |U_x| - 1$ such that $u \in U_x^i$, a child node y is created (see Algorithm 12). For each such node y, $F_y \leftarrow F_x$, $U_y \leftarrow U_x$, $CandUpd_y \leftarrow \emptyset$ and $U_y^0 \leftarrow U_y^0 \setminus \{u\}$. Now, if $i = 0$, a new s-club cluster is added to node y: $F_y \leftarrow F_y \cup \{F_y^l\}$, $U_y \leftarrow U_y \cup \{U_y^l\}$, where $l = |F_x|$, $F_y^l = \{u\}$, and $U_y^l = U_y^0$. Additionally, set U_y^l is recorded in $CandUpd_y$ to be updated later during the processing of node y (see Algorithm 13). Otherwise, if $i \geq 1$, then vertex u is simply added to set F_y^i. Finally, for each $j = 1, \ldots, |U_x| - 1$ such that $u \in U_x^j$, branching vertex u is removed from the corresponding U_y^j. Every such affected set U_y^j is then recorded in $CandUpd_y$ as it will need to be updated later when node y is processed (see Algorithm 13).

After branching, a parent node x will have as many children nodes as the number

Algorithm 11 A branch-and-bound algorithm (BB) for minimum s-club partitioning

Input: $G = (V, E)$
Output: A minimum s-club partitioning $\mathcal{P}_s^*(G)$

$\mathcal{P}_s^*(G) \leftarrow$ initial heuristic solution (Section 4.2.3), $UB \leftarrow |\mathcal{P}_s^*(G)|$
$U_x^0 \leftarrow V, F_x^0 \leftarrow \emptyset, U_x \leftarrow \{U_x^0\}, F_x \leftarrow \{F_x^0\}$
$CandUpd_x \leftarrow \emptyset$ ▷ Set of candidate sets to be updated in PROCESS(x)
$x \leftarrow \{F_x, U_x, CandUpd_x\}, Tree \leftarrow x$ ▷ Create root node x and initialize the BB tree
while $Tree \neq \emptyset$ **do**
  Pick a node $x \in Tree$ using depth-first search strategy
  $Tree \leftarrow Tree \setminus \{x\}$
  PROCESS(x)
  if at least one F_x^i is not an s-clique in $G[F_x^i \cup U_x^i]$ for some $i = 1, \ldots |F_x| - 1$ **then**
    Fathom x by infeasibility
  else
    $LB_x \leftarrow$ lower bound (Section 4.2.4)
    if $LB_x \geq UB$ **then** fathom x by bound
    else
      if sets $U_x^0 \setminus (\bigcup_{i=1}^{|U_x|-1} U_x^i)$ and $F_x^i \cup U_x^i$ for all $i = 1, ..., |U_x| - 1$ form a feasible s-club partitioning of G **then**
        if $U_x^0 \setminus (\bigcup_{i=1}^{|U_x|-1} U_x^i) \neq \emptyset$ **then**
          $\mathcal{P}_s^*(G) \leftarrow \{U_x^0 \setminus (\bigcup_{i=1}^{|U_x|-1} U_x^i)\} \cup (\bigcup_{i=1}^{|U_x|-1} \{F_x^i \cup U_x^i\})$
          $UB \leftarrow |U_x|$
        else
          $\mathcal{P}_s^*(G) \leftarrow \bigcup_{i=1}^{|U_x|-1} \{F_x^i \cup U_x^i\}$
          $UB \leftarrow |U_x| - 1$
        Fathom x by feasibility
      else
        BRANCH($x, Tree$)
return $\mathcal{P}_s^*(G)$

of sets $U_x^i, i = 0, \ldots, |U_x| - 1$ that contain the branching vertex u. To create a smaller number of branches at the top layers of the BB tree, we will use a branching rule that selects a vertex belonging to the smallest number of candidate sets in the current

Algorithm 12 Branching procedure in BB algorithm

procedure BRANCH($x, Tree$)
  Pick $u \in U_x^0$ that belongs to the smallest number of sets $U_x^i, i = 0, \ldots, |U_x| - 1$
  for each $i = 0, \ldots, |U_x| - 1$ such that $u \in U_x^i$ **do**
    $F_y \leftarrow F_x, U_y \leftarrow U_x, CandUpd_y \leftarrow \emptyset$ ▷ Start creating a child node y
    $U_y^0 \leftarrow U_y^0 \setminus \{u\}$
    if $i = 0$ **then**
      $l \leftarrow |F_x|$ ▷ Initialize a new s-club cluster for child node y
      $F_y^l \leftarrow \{u\}$
      $U_y^l \leftarrow U_y^0$
      $CandUpd_y \leftarrow \{U_y^l\}$
      $F_y \leftarrow F_y \cup \{F_y^l\}, U_y \leftarrow U_y \cup \{U_y^l\}$
    else
      $F_y^i \leftarrow F_y^i \cup \{u\}$ ▷ Update i-th s-club cluster of y
    for each $j = 1, \ldots, |U_x| - 1$ such that $u \in U_x^j$ **do**
      $U_y^j \leftarrow U_y^j \setminus \{u\}$ ▷ Update all candidate sets that contain u
      $CandUpd_y \leftarrow CandUpd_y \cup \{U_y^j\}$ ▷ Record candidate sets to be further updated
    $y \leftarrow \{F_y, U_y, CandUpd_y\}$
    $Tree \leftarrow Tree \cup \{y\}$

BB tree node.

To encourage detection of feasible solutions at the earlier stages of the BB algorithm, depth-first search (DFS) strategy is chosen to select the search tree nodes for further processing. During node x processing, each set U_x^i that had been recorded in $CandUpd_x$ upon creation of node x is updated. This updating procedure is summarized in Algorithm 13 and consists of iterative removal of all vertices in each $U_x^i \in CandUpd_x$ that are at distance of more than s from at least one vertex of the corresponding F_x^i in $G[F_x^i \cup U_x^i]$.

When node x processing is complete, if there is at least one set F_x^i for some $i = 1, \ldots, |F_x| - 1$ that does not form an s-clique in $G[F_x^i \cup U_x^i]$, then BB node x is fathomed by *infeasibility*. Otherwise, a lower bound on the size of a minimum s-club

Algorithm 13 Processing a node of the search tree in BB algorithm

procedure PROCESS(x)
  for each $i = 1, \dots |U_x| - 1$ such that $U_x^i \in CandUp_x$ **do**
    $U_{Drop} \leftarrow \{u \in U_x^i$: there exists $v \in F_x^i$ such that $d_{G[F_x^i \cup U_x^i]}(u, v) >s\}$
    while $U_{Drop} \neq \emptyset$ **do**
      $U_x^i \leftarrow U_x^i \setminus U_{Drop}$
      $U_{Drop} \leftarrow \{u \in U_x^i$: there exists $v \in F_x^i$ such that $d_{G[F_x^i \cup U_x^i]}(u, v) >s\}$

partitioning with the current partial assignment is estimated (see Section 4.2.4). If this bound is not smaller than the size of the incumbent solution, then the BB node is fathomed by *bound*. If BB node x is neither fathomed by infeasibility nor by bound, then we check and see if sets $U_x^0 \setminus (\bigcup_{i=1}^{|U_x|-1} U_x^i)$ and $F_x^i \cup U_x^i$ for all $i = 1, ..., |U_x| - 1$ form a feasible s-club partitioning of G. If yes, then node x is fathomed by *feasibility* and the incumbent solution is updated. Otherwise, branching procedure is called and new children nodes are added to the BB search tree.

4.2.3 Upper bound heuristics

To construct an initial feasible s-club partitioning, the following iterative heuristics can be utilized based on two well-known heuristics for constructing s-clubs, namely DROP and CONSTELLATION [11]:

DROP-based heuristic

DROP heuristic was designed by Bourjolly et al. [11] to find large s-clubs in a given graph. DROP takes an initial graph as an input and iteratively removes from it vertices with the smallest number of s-neighbors (vertices located at distance of at most s from a vertex) until an s-club is obtained. When an s-club is found, DROP can be repeatedly applied to the subgraph induced by the remaining vertices not included in the previously detected s-clubs yielding

a collection of mutually non-overlapping (partitioning) s-clubs.

CONSTELLATION-based heuristic

CONSTELLATION is another heuristic by Bourjolly et al. [11] for identifying large s-clubs. This technique is based on the fact that the star graph is a 2-club and is generalized for s-club detection by iteratively building a star graph on one of the vertices of an existing star graph until the total number of star graphs built is equal to $s-1$. At each iteration of this heuristic, the star graph is built on the vertex with the largest number of adjacent vertices not already included in the solution. Similarly to DROP-based technique above, CONSTELLATION can be applied iteratively to partition an initial graph into s-clubs.

To initialize the incumbent solution in Algorithm 11, we employ both DROP- and CONSTELLATION-based heuristics and select the solution with the smaller number of partitions among the obtained solutions. This procedure is referred to as the Upper Bound Heuristic (UBH) in the remainder of this chapter.

4.2.4 Lower bound estimation

To estimate a lower bound of the solution, we use the fact that the size of any s-independent set in G is a lower bound on the size of a minimum s-club partitioning in this graph. For a graph $G=(V,E)$, a set $I^s \subseteq V$ is called an s-independent set in G if the distance between any two vertices $u, v \in I^s$ in G is at least $s+1$, i.e., $d_G(u,v) > s$. It is easy to verify that no two vertices in an s-independent set in G can belong to the same s-club in this graph. Thus, the cardinality of an s-independent set in G is a lower bound on the number of s-club partitions required to span all vertices of this graph.

The lower bound at each node x of BB tree can be derived in the following manner. Let I_G^s be an s-independent set in the original graph G, and I_x^s be an s-independent set in $G[U_x^0]$ formed by vertices from $U_x^* = U_x^0 \setminus (\bigcup_{i=1}^{|U_x|-1} U_x^i)$. As mentioned earlier, $|I_G^s|$ is a lower bound on the size of a minimum s-club partitioning of G, which is globally valid for all search tree nodes of BB. On the other hand, the following observations are valid for any feasible s-club partitioning of G, say $\mathcal{P}_s(G)$, obtained in the subtree rooted at the present tree node x. First, none of the elements of U_x^* can belong to any of the s-club partitions in $\mathcal{P}_s(G)$ that contain one of the fixed sets $F_x^i, i = 1, \ldots, |F_x| - 1$. Second, since I_x^s forms an s-independent set in $G[U_x^0]$, none of two vertices from this set can belong to the same s-club partition in $\mathcal{P}_s(G)$. Therefore, $|\mathcal{P}_s(G)| \geq |I_x^s| + |F_x| - 1$. As a result,

$$LB_x = \max\{|I_G^s|, |I_x^s| + |F_x| - 1\}$$

is a valid lower bound on the size of any feasible solution obtained in the subtree rooted at node x.

To find I_G^s at the start of the BB algorithm, a simple greedy heuristic is employed that results in a maximal (not contained in a larger one) s-independent set in G. First, we construct the s-th power graph of G. Given $G = (V, E)$, the s-th power graph is defined as $G^s = (V, E^s)$, where $E^s = \{(u, v) : u, v \in V; d_G(u, v) \leq s\}$. Any independent set in G^s corresponds to an s-independent set in G, and vice versa. Hence, we can utilize a simple greedy heuristic for constructing a maximal independent set and apply it to G^s. Such a heuristic constructs a solution iteratively. First element to be added to the solution is the vertex with the smallest number of adjacent vertices. During each subsequent iteration, a vertex not adjacent to any of the vertices already in the solution and with the smallest number of adjacent vertices

that are not adjacent to any vertex in the solution is selected and placed in the solution. This procedure continues until no more vertices can be added to the solution set.

At each node x of the BB tree, the greedy heuristic mentioned above is also used to obtain a maximal s-independent set in $G[U_x^0]$ that is a subset of U_x^* and can be used as set I_x^s in our lower bound estimation.

4.3 A mixed 0-1 linear programming formulation

Below, we formulate the minimum s-club partitioning problem as a mixed 0-1 linear program. Let z define the number of s-clubs needed to partition the graph and

$$x_{vk} = \begin{cases} 1 & \text{if vertex } v \text{ belongs to the } k\text{-th } s\text{-club,} \\ 0 & \text{otherwise.} \end{cases}$$

Additionally, let P_{uv} denote the set of all indexed paths of length at most s between vertices u and v, p_{uv}^t - the t-th path in P_{uv}, $V(p_{uv}^t)$ - the set of vertices on path p_{uv}^t, and let y_{uv}^t be an auxiliary binary variable corresponding to every path $p_{uv}^t \in P_{uv}$.

The s-club partitioning problem can now be formulated as follows:

$$\min \quad z \tag{4.1a}$$

$$\text{s.t.} \quad z \geq k x_{vk} \qquad \forall v \in V, k = 1, \ldots, K, \tag{4.1b}$$

$$\sum_{k=1}^{K} x_{vk} = 1 \qquad \forall v \in V. \tag{4.1c}$$

$$x_{uk} + x_{vk} - \sum_{t: p_{uv}^t \in P_{uv}} y_{uv}^t \leq 1 \qquad \forall (u,v) \notin E, k = 1, \ldots, K, \tag{4.1d}$$

$$1 - x_{uk} + x_{rk} \geq y_{uv}^t \qquad \forall r \in V(p_{uv}^t), p_{uv}^t \in P_{uv}, (u,v) \notin E, k = 1, \ldots, K, \tag{4.1e}$$

$$x_{vk} \in \{0,1\} \qquad \forall v \in V, k = 1, \ldots, K, \tag{4.1f}$$

$$y_{uv}^t \in \{0,1\} \qquad \forall p_{uv}^t \in P_{uv}, (u,v) \notin E, \tag{4.1g}$$

$$z \geq 0. \tag{4.1h}$$

Note that K is a pre-calculated upper bound on the number of s-club clusters in G. Obviously, $K \leq |V|$. Constraints (4.1c) ensure that each vertex belongs to a single cluster only. The s-club structure of the clusters is enforced by the constraints (4.1d) and (4.1e). More precisely, constraints (4.1d) ensure that there is at least a path of length at most s between any pair of vertices within the same cluster, and constraints (4.1e) guarantee that all vertices on one such path belong to the same cluster. The formulation contains a large number of binary variables and constraints. Therefore, it is beneficial to have K as small as possible. One can use a heuristic approach to find a feasible partition, and set K to the achieved number of clusters.

In our numerical experiments presented in Section 4.4, we focus on the special case of the minimum s-club partitioning problem with $s = 2$. For this special case, formulation (4.1) can be significantly simplified. Variables y_{uv}^t can be eliminated, and (4.1) reduces to

$$\min \quad z \tag{4.2a}$$

$$\text{s.t.} \quad z \geq k x_{vk} \qquad \forall v \in V, k = 1, \ldots, K, \tag{4.2b}$$

$$\sum_{k=1}^{K} x_{vk} = 1 \qquad \forall v \in V, \tag{4.2c}$$

$$x_{uk} + x_{vk} - \sum_{\forall w \in N(u) \cap N(v)} x_{wk} \leq 1 \qquad \forall (u, v) \notin E, k = 1, \ldots, K, \tag{4.2d}$$

$$x_{vk} \in \{0, 1\} \qquad \forall v \in V, k = 1, \ldots, K, \tag{4.2e}$$

$$z \geq 0. \tag{4.2f}$$

Here, constraints (4.2d) ensure that all the vertices in each cluster k form a 2-club.

4.4 Computational experiments

In this section, we report results of computational experiments for the special case of the minimum s-club partitioning problem with $s = 2$ to test performance of the proposed algorithms in this chapter. The experiments are conducted on a set of selected real-life graphs from 10^{th} DIMACS Implementation Challenge [23] and randomly generated graph instances from [33]. The latter ones had been originally generated and used to test the performance of the exact combinatorial branch-and-bound algorithms for the maximum s-club problem [52, 67]. These graphs are generated using two parameters, a and b, where $0 \leq a \leq b \leq 1$ and the value of $(a + b)/2$ determines the graph's expected edge density, and $b - a$ - its *vertex degree variance*. For our experiments, we consider graphs with the minimum vertex degree variance (achieved when $a = b$), on 100, 150, and 200 vertices, and with expected edge densities equal to 0.0125, 0.025, 0.05, 0.1, 0.15 and 0.2. Ten randomly generated instances for each combination of mentioned vertex set size and edge density are used, but only

the averages of the results among ten are reported.

Both solution techniques, the branch-and-bound (BB) algorithm and the mixed 0-1 linear programming (M01P) formulation, are coded in C++ and compiled using Microsoft Visual Studio 2010. Gurobi Optimizer 6.5.1 [37] is used as the solver for the mixed 0-1 linear programming formulation. The experiments are conducted on a machine with Windows 7 Enterprise OS, 2.40 GHz CPU and 12 GB RAM. The results of the experiments are summarized in Tables 4.1 - 4.4.

Tables 4.1 and 4.2 are devoted to the real-life graphs. First three columns of Table 4.1 contain general information about the instance: its name, size of its vertex set ($|V|$) and its edge set ($|E|$). The fourth column of Table 4.1 reports the size of the initial solution detected using the upper bound heuristic (Section 4.2.3) during the initialization phase of the BB algorithm. Next two sets of three columns of Table 4.1 include information regarding the performance of the BB algorithm and M01P, respectively. Namely, the size of the best detected solution, the optimality gap and the running time of the algorithms are reported. The optimality gap is computed as a ratio of the difference between the size of the best found solution and the size of the lower bound to the size of the best found solution, and converted into a percentage.

Note that for M01P, parameter K from Formulation (4.2) is set to the size of the initial solution detected using UBH during the initialization phase of the BB algorithm. Hence, the running time reported for M01P also includes the running time of UBH algorithm. Additionally, the running time limit for the BB algorithm is set to 3600 seconds, and for M01P to 3600 seconds minus the running time of UBH during BB implementation. The decision to set such a time limit for M01P was made to ensure a fair comparison of BB and M01P, although, for most of the cases, the running time of UBH has shown to be negligible (less than a second to a

Table 4.1: Results of the experiments on a set of real-life instances. UBH is the size of the initial solution obtained by the upper bound heuristic. The size of the best solution found (Size), optimality gap (Gap(%)), and running time (CPU(s)) for both BB and M01P algorithms are also reported.

Instance	$\|V\|$	$\|E\|$	UBH	**BB**			**M01P**		
				Size	Gap(%)	CPU(s)	Size	Gap(%)	CPU(s)
karate	34	78	4	4	0	0.01	4	0	0.06
chesapeake	39	170	3	3	0	0.01	3	0	0.05
dolphins	62	159	19	13	0	0.53	13	0	71.79
lesmis	77	254	23	10	0	0.22	10	0	13.01
polbooks	105	441	18	18	33.33	3600.01	12	0	1005.36
adjnoun	112	425	33	18	0	6.70	18	72.22	3600.02
football	115	613	14	10	0	616.69	11	36.36	3602.90
jazz	198	2742	16	16	18.75	3600.19	13	76.92	3599.51
CN[1]	297	2148	27	27	51.85	3600.00	27	96.30	3600.18
CM[2]	453	2025	88	29	0	42.13	88	-	3600.75
email	1133	5451	347	212	2.83	3603.08	347	-	-
polblogs	1490	16715	566	566	30.39	4086.50	566	-	-
netscience	1589	2742	504	481	0.83	3600.31	504	-	-
data	2851	15093	356	272	10.29	3600.22	356	-	-

[1]celegansneural
[2]celegans_metabolic

few seconds). The total elapsed time of BB is monitored after each BB search tree node is processed, and if this time exceeds the limit, the algorithm terminates. As a result, the running times reported for BB might be larger than 3600 seconds for some instances. For M01P the time limit is enforced by setting internal parameter of Gurobi Optimizer to a desired level.

First three columns in Table 4.2 are identical to the ones in Table 4.1. Columns 4 though 7 contain some additional details of the BB algorithm: the total number of BB search tree nodes (# Nd), the number of BB search tree nodes fathomed by feasibility (# FFea), infeasibility (# FInfea), and bound (# FBou). The last column of Table 4.2 contains details of M01P algorithm, i.e., the total number of explored

Table 4.2: Details of the experiments on a set of real-life instances. Number of search tree nodes (# Nd), number of nodes fathomed by feasibility (# FFea), infeasibility (# FInfea), and bound (# FBou) in BB algorithm are reported. For M01P algorithm, # Nd reports the number of search tree nodes in this algorithm.

Instance	$\|V\|$	$\|E\|$	**BB**				**M01P**
			# Nd	# FFea	# FInfea	# FBou	# Nd
karate	34	78	1	0	0	1	0
chesapeake	39	170	1	0	0	1	0
dolphins	62	159	640	2	21	330	0
lesmis	77	254	203	1	9	116	0
polbooks	105	441	7764045	0	6401143	52132	0
adjnoun	112	425	1340	1	28	691	0
football	115	613	196604	4	4923	97722	1703
jazz	198	2742	809790	0	690265	3191	0
CN[1]	297	2148	420180	0	343427	1	0
CM[2]	453	2025	1571	1	90	1027	0
email	1133	5451	21300	1	367	11153	-
polblogs	1490	16715	1090	0	1	0	-
netscience	1589	2742	43810	1	33	21183	-
data	2851	15093	33455	1	124	15137	-

[1]celegansneural
[2]celegans_metabolic

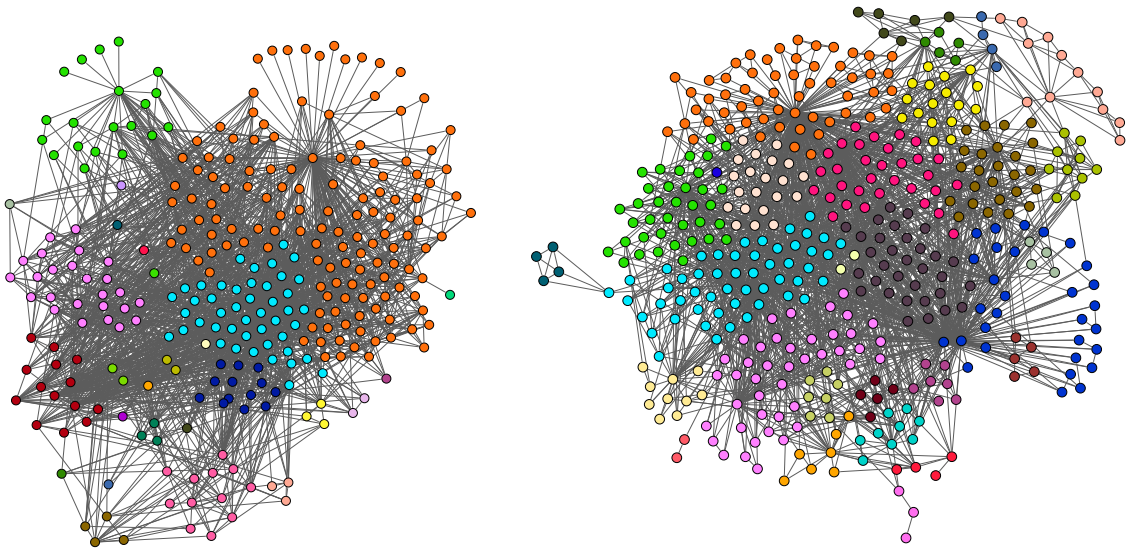
nodes (# Nd) in this algorithm.

As it can be seen in Table 4.1, the BB algorithm detected an optimal solution in half of the real-life instances, however, mostly small-sized. Among the other half not being solved to optimality, BB achieved an improvement from the initial feasible solution obtained by UBH in a little less than a half of instances, mostly large-scaled. Based on optimality gap and running time, we can also argue that BB has shown superior performance on this set of graphs compared to M01P. Except one instance (i.e., polbooks), BB either solves the problem to optimality much faster than M01P, or returns a smaller optimality gap in case of reaching the running time limit. For one instance (i.e., celegans_metabolic), M01P could not even report an optimality gap as

it could not solve the linear programming relaxation of Formulation (4.2) in the root node of the search tree within the time limit and failed to find a valid lower bound. Furthermore, M01P failed in four of the largest instances, as Formulation (4.2) could not even be loaded into the machine's memory, while BB reported nontrivial optimality gaps for these instances. Note that for these instances, we report the size of the initial feasible solution obtained by UBH as the size of the best solution found by M01P, because we used this solution to find parameter K and construct Formulation (2).

According to Table 4.2, small number of tree nodes fathomed by feasibility in BB algorithm indicates that partitioning these instances into 2-club clusters is a challenging task. On the other hand, the values reported for the number of nodes fathomed by infeasibility and bound show the effectiveness of these fathoming approaches, especially our proposed lower bounding scheme. Regarding M01P algorithm, for all instances that could be loaded into the machine's memory (except one, i.e., football), the branching did not even start within the considered running time limit and the reported results are all obtained in the root node of the search tree.

Illustrated in Fig. 4.2 are solutions obtained by BB for two biological networks, celegansneural and celegans_metabolic. For celegansneural, the best found solution is demonstrated (partitioning of the graph into 27 2-club clusters), while for celegans_metabolic the optimal solution is presented (partitioning into 29 2-club clusters). For each graph, the vertices belonging to the same partition (2-club) are shown by the same color. According to this figure, it appears that most of these 2-club clusters have a single-vertex kernel and are extremely sparse. This further suggests the central role of these kernels in connecting the other members of each cluster and might have significance in understanding the functions of the underlying biological systems.



(a) Celegansneural (b) Celegans_metabolic

Figure 4.2: 2-club partitioning of the graphs representing biological networks.

The performance of BB and M01P on randomly generated graphs is reported in Tables 4.3 and 4.4. Here, the general information about each instance is included in the first two columns. Namely, the size of the vertex set and the expected edge density are reported. The rest of the columns in Tables 4.3 and 4.4 contain the same information as in Tables 4.1 and 4.2, except, each column reports the average measure among the ten instances tested, and Table 3 has an extra column (# NoSol) that reports the number of graph instances for which no feasible solution could be detected by M01P within the set time limit. For such instances, we report the size of the initial feasible solution obtained by UBH as the size of the best solution found by M01P and use it to compute the optimality gap.

According to Table 4.3, BB shows a similar dominance over M01P on randomly generated graphs of small edge densities. More specifically, BB outperforms M01P in terms of optimality gap on all instances of densities 0.0125, 0.025, and 0.05. As edge density increases, it appears that M01P algorithm shows a better computational

Table 4.3: Results of the experiments on randomly generated instances. UBH is the average size of the initial solution obtained by the upper bound heuristic. The average size of the best solution found (Size), average optimality gap (Gap(%)), and average running time (CPU(s)) for both BB and M01P algorithms are also reported. For M01P algorithm, # NoSol reports the number of instances for which no feasible solution was obtained by M01P within the time limit.

$\|V\|$	Density	UBH	**BB**			**M01P**			
			Size	Gap(%)	CPU(s)	Size	Gap(%)	CPU(s)	# NoSol
100	0.0125	58.6	57.2	0	0.17	57.2	57.92	3450.19	0
	0.025	40.9	35.8	0	6.09	36.0	81.22	3600.68	0
	0.05	27.8	22.2	11.15	1105.10	20.7	42.26	3600.25	0
	0.1	18.0	18.0	58.89	3600.01	13.6	40.36	3600.65	0
	0.15	12.8	12.8	70.99	3600.04	8.4	42.30	3600.64	0
	0.2	9.1	9.1	71.14	3600.12	4.0	2.00	887.42	0
150	0.0125	73.2	68.0	0	0.58	69.2	97.10	3600.20	0
	0.025	50.0	41.1	6.01	2453.91	46.9	95.72	3600.25	0
	0.05	32.8	30.8	45.88	3600.01	31.5	93.64	3600.10	0
	0.1	19.4	19.4	70.15	3600.02	18.2	83.51	3599.91	0
	0.15	14.4	14.4	78.99	3600.24	12.3	78.99	3599.67	0
	0.2	7.3	7.3	72.07	3600.35	2.4	0	581.01	0
200	0.0125	83.1	73.4	1.35	2669.22	81.3	97.78	3600.19	0
	0.025	57.1	47.8	21.05	3600.20	55.4	97.46	3600.22	1
	0.05	35.0	35.0	58.88	3600.01	35.0	94.86	3600.07	4
	0.1	21.5	21.5	77.42	3600.12	20.4	85.23	3599.48	2
	0.15	15.7	15.7	83.88	3600.66	15.7	92.85	3598.93	1
	0.2	4.6	4.6	54.67	3600.12	2.0	0	45.33	0

performance compared to BB algorithm. In terms of optimality gap, M01P performs better than BB on 100-vertex graphs of densities larger than 0.05, and on 150- and 200-vertex graphs of density 0.2. Specifically for the density of 0.2, M01P performs extremely well, finding optimal solution for most of the 100-vertex instances and for all of the 150- and 200-vertex instances. Such drastic improvement in M01P performance can be justified by the fact that as the input graph edge density increases, the number of constraints (4.2d) decreases in Formulation (4.2) making it easier to solve.

Table 4.4: Details of the experiments on randomly generated instances. Average number of search tree nodes (# Nd), average number of nodes fathomed by feasibility (# FFea), infeasibility (# FInfea), and bound (# FBou) in BB algorithm are reported. For M01P algorithm, # Nd reports the average number of search tree nodes in this algorithm.

$\|V\|$	Density	**BB**				**M01P**
		# Nd	# FFea	# FInfea	# FBou	# Nd
100	0.0125	143.5	0.9	2.5	49.5	7956.8
	0.025	5562.8	1.6	137.1	2566.4	5651.9
	0.05	4272839.3	1.9	3107301.0	54512.0	2658.7
	0.1	7869391.0	0	5390474.2	473354.1	3438.3
	0.15	3208235.5	0	2120720.2	236004.0	10826.1
	0.2	1215940.0	0	923566.8	36772.8	28786.1
150	0.0125	307.5	1.2	9.4	130.7	0
	0.025	3148258.0	3.2	1861418.7	378047.7	0
	0.05	9945102.0	0.6	7688678.1	272026.2	0
	0.1	1767993.0	0	936068.6	263470.7	0
	0.15	317570.5	0	255482.8	4028.0	0
	0.2	494123.0	0	382342.8	12073.5	3540.8
200	0.0125	851210.9	2.5	28402.2	381662.5	0
	0.025	3188227.5	2.4	1881298.0	389878.7	0
	0.05	4558097.0	0	3222895.4	234654.2	0
	0.1	452740.5	0	257738.3	61581.4	0
	0.15	98666.5	0	82145.2	19.5	0
	0.2	296235.0	0	183621.4	24620.5	0

We also observe from # NoSol values that M01P was not able to attain a feasible solution within the time limit for some of the 200-vertex instances of densities from 0.025 to 0.15. This once again shows that as the size of the graph increases for such densities, it becomes harder to detect a feasible solution by M01P.

Based on Table 4.4, similar to the observations made for Table 4.2, small number of tree nodes are fathomed by feasibility in BB algorithm. This again shows the difficulty associated with finding a feasible 2-club partitioning in these instances.

Moreover, the number of nodes fathomed by infeasibility and bound in BB algorithm once again indicate the effectiveness of these fathoming schemes. Considering M01P algorithm, for all 150- and 200-vertex instances (except 150-vertex instances with density 0.2), the branching in the search tree did not even start within the given running time limit and the reported results are all obtained in the root node.

We also observe in Table 4.3 that the average size of the 2-club partitioning for graphs with edge density of 0.2 is decreasing as the size of the graphs is increasing. To explain such abnormality, first note that for these particular instances, the average size of the largest 2-club in 100-vertex graphs is 72.5, in 150-vertex graphs - 123.6, and in 200-vertex graphs - 178.2 [52]. Therefore, as the size of these graphs increases from 100 to 200, it appears that they tend to contain larger 2-club clusters relative to the size of the original graph and as a result, they can be partitioned into smaller number of such clusters. Note that in general, such behavior is not expected for sparse graphs, because the size of the partitioning in such instances will most likely increase as the size of the input graph increases.

In summary, the results of our experiments show that even though M01P performs well on some relatively dense instances, the branch-and-bound algorithm is still a better alternative for the real-life instances as such graphs tend to have smaller edge densities and are usually large-scaled.

5. CONCLUSION AND FUTURE WORK

Analysis of complex systems using graph theory tools, or also known as network analysis, has been gaining a lot of interest among a wide variety of disciplines including social science, computer science, biology, genetics, transportation, telecommunication. Graphs provide an intuitive and succinct representation of massive data extracted from the underlying system. Finding large tight clusters and partitioning the data into cohesive subgroups are common analysis techniques that can help draw important information about the underlying system, as well as help reduce the size of the original data set. Since the first model of a tight cluster, a clique, was introduced in 1949, many other formalizations of a cluster, less restrictive than the clique, have been proposed and have found important applications in various fields. Cluster models that ensure low diameter, referred to as s-clubs with 1-club being clique, are especially worth attention, as they naturally arise in so many real-life systems, especially in transportation, telecommunication, and even biological networks.

In this dissertation, our contribution to the field of network analysis comprised of studying the properties of low-diameter clusters, introducing new extensions to address network vulnerability to failures and uncertainties, and designing solution techniques to solve related optimization problems, including the problem of finding low-diameter clusters of the largest size and the problem of decomposing the network into the minimum number of non-overlapping low-diameter clusters.

In Chapter 2, we introduced a notion of a robust clique in uncertain graphs. More specifically, we considered graphs subject to uncertain multiple edge failures and studied a problem of detecting robust cliques in such graphs. To formalize the losses associated with the edge failures, we defined a loss function that controls the

maximum number of neighbors that any vertex in the clique can lose as a result of edge failures. As the measure to manage the loss, we used a well-known and intuitive risk metric – the Conditional Value-at-Risk that is advantageous in several respects. Due to the computational intractability of the problem of interest, we tackled the problem by heuristic approach. Namely, we developed three metaheuristics, which are adaptations of the well-known tabu search, its variant STABULUS, and GRASP, respectively. The main feature of all three algorithms is the selection rule for the vertices to be added (deleted, swapped) to (from) the solution. As computing or estimating the value of CVaR is computationally challenging, we proposed to use a probability of the largest loss for an individual vertex as a part of the criterion to select the right candidate vertex to be added, deleted or swapped. To analyze the heuristic's effectiveness we conducted computational experiments on a set of DIMACS graph instances, and also computed exact solutions using a modification of Östergård's algorithm for smaller graphs. The heuristics outperformed the exact algorithm both in terms of running time and the quality of the solution within the allowed timeframe, and proved to be an effective tool for detecting robust cliques.

When studying robust cliques we restricted our work and experiments to one specific loss function and CVaR risk measure. Studying other types of losses that yield other clique relaxations in the post-failure graph, such as s-defective clique or s-club could be a fruitful future research direction. Moreover, considering other risk measures could be an interesting extension to this work.

In Chapter 3, we focused on graphs where uncertainty is not explicitly defined and considered the special case of diameter-based relaxation of a clique, a 2-club. This model provides a much less costly alternative to a clique, but can be quite vulnerable to network disruptions such as node failures. The robustness of 2-club to such disruptions can be mitigated by imposing additional connectivity requirements

to ensure if one of the critical nodes fails, the elements of a cluster are still connected. Such a connectivity requirement is especially important in transportation and communication networks. By studying the connectivity properties of 2-clubs, we observed that a given 2-club is not biconnected (fragile) if and only if there exists a unique hub vertex in the 2-club that is also the only cut vertex in the corresponding induced subgraph. This property provides a fast verification of any 2-club's biconnectivity, and a polynomial-time framework for detecting a maximum fragile 2-club. Consequently, we designed a polynomial-time solution procedure for the maximum fragile 2-club. For the maximum biconnected 2-club problem, we modified an existing combinatorial branch-and-bound algorithm by incorporating the biconnectivity verification procedure, and also designed a branch-and-cut approach utilizing a linear 0-1 program with nontrivial biconnectivity constraints applied in a lazy fashion. The results of computational experiments indicated that for relatively dense graphs a maximum biconnected 2-club can be found by the combinatorial branch-and-bound algorithm within the same amount of time as spent on detecting a maximum 2-club without any considerations on its connectivity. Moreover, the branch-and-cut approach on the same set of instances has shown even better performance, but did not prove to dominate the combinatorial branch-and-bound algorithm on large real-life graphs. The experiments also showed that the largest 2-clubs are likely to be fragile in sparse graphs, and biconnected in dense ones.

In future work, it would be beneficial to discover facet inducing inequalities and effective cuts for the maximum biconnected 2-club problem. Studying other special cases of k-connected s-clubs, as well as the k-hereditary s-club model mentioned in Chapter 3, are two additional interesting directions for future research.

Lastly, we also looked into the problem of partitioning the graph into s-clubs. Given the abundance of various graph-based clustering techniques available in the

literature, it was to our surprise that very little research has been done in the area of partitioning the graph into clique relaxations. We studied the minimum s-club partitioning problem for the general class of simple undirected graphs by formulating it using a mixed 0-1 linear program and designing an exact combinatorial branch-and-bound algorithm to solve this problem. The results of our numerical experiments for the special case of $s = 2$ have shown that the proposed combinatorial branch-and-bound algorithm performs better on real-life instances and on large instances with smaller edge densities. On the other hand, solution of the proposed mixed 0-1 linear program by branch-and-cut algorithms available via commercial solvers might be a better approach on smaller graphs of higher densities. According to our numerical experiments, this problem is very hard to solve in practice, and some even moderately-sized graphs could not be solved to optimality within 1 hour limit. Perhaps, some more sophisticated heuristic or metaheuristic approaches could be designed either as a stand-alone solution technique or as an upper bound method within a branch-and-bound algorithm. A tighter lower bound estimation could also drastically improve the performance of a branch-and-bound algorithm.

Partitioning into other types of clique relaxations might be more suitable in some other applications making this problem an interesting future research direction. Detection of largest clique relaxations of various types has been broadly studied, both theoretically and algorithmically, however, almost no structured research is done on the problems of finding a minimum partitioning of the graph into such clique relaxations. Furthermore, in some cases, it may even be more applicable to consider a partitioning where each partition could form a different clique relaxation, e.g., one is an s-club, the other one is an s-plex, and so on. Such approach could be beneficial in applications where no specific uniform structural requirement is imposed.

REFERENCES

[1] N. Abbas and L. Stewart. Clustering bipartite and chordal graphs: Complexity, sequential and parallel algorithms. *Discrete applied mathematics*, 91(1):1–23, 1999.

[2] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer communications*, 30(14):2826–2841, 2007.

[3] J. Abello, P. M. Pardalos, and M. G. C. Resende. On maximum clique problems in very large graphs. *DIMACS series*, 50:119–130, 1999.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[5] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[6] E. Balas and J. Xue. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica*, 15:397–412, 1996.

[7] B. Balasundaram and S. Butenko. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*, pages 865–890. Springer, 2006.

[8] B. Balasundaram, S. Butenko, and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005.

[9] V. L. Boginski, C. W. Commander, and T. Turko. Polynomial-time identification of robust network flows under uncertain arc failures. *Optimization Letters*, pages 461–473, 3 2009.

[10] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Springer US, 1999.

[11] J.-M. Bourjolly, G. Laporte, and G. Pesant. Heuristics for finding k-clubs in an undirected graph. *Computers & Operations Research*, 27(6):559–569, 2000.

[12] J.-M. Bourjolly, G. Laporte, and G. Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, 2002.

[13] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[14] S. Butenko and W. E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.

[15] R. Carraghan and P. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[16] J.-M. Chang, J.-S. Yang, and S.-L. Peng. On the complexity of graph clustering with bounded diameter. In *Computer Science and Engineering Conference (ICSEC), 2014 International*, pages 18–22. IEEE, 2014.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[18] F. Corno, P. Prinetto, and M. Sonza Reorda. Using symbolic techniques to find the maximum clique in very large sparse graphs. In *Proceedings of the 1995 European conference on Design and Test*, EDTC '95, pages 320–324, Washington, DC, USA, 1995. IEEE Computer Society.

[19] T. De, A. Pal, and I. Sengupta. Traffic grooming, routing, and wavelength assignment in an optical wdm mesh networks based on clique partitioning. *Photonic Network Communications*, 20(2):101–112, 2010.

[20] J. S. Deogun, D. Kratsch, and G. Steiner. An approximation algorithm for clustering graphs with dominating diametral path. *Information Processing Letters*, 61(3):121–127, 1997.

[21] R. Diestel. *Graph Theory*. Springer Heidelberg, 2005.

[22] DIMACS. NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. The Second DIMACS Implementation Challenge. http://dimacs.rutgers.edu/Challenges/, 1992-1993.

[23] DIMACS. Algorithm Implementation Challenge: Graph Partitioning and Graph Clustering. The Tenth DIMACS Implementation Challenge. http://dimacs.rutgers.edu/Challenges/, 2012.

[24] U. Dorndorf, F. Jaehn, and E. Pesch. Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*, 42(3):292–301, 2008.

[25] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.

[26] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[27] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[28] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 31–37. ACM, 2002.

[29] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[30] C. Friden, A. Hertz, and D. de Werra. STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.

[31] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman New York, 1979.

[32] M. Gendreau, P. Soriano, and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41:385–403, 1993.

[33] M. Gendreau, P. Soriano, and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41(4):385–403, 1993.

[34] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[35] F. Glover. Tabu search. Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[36] F. Glover. Tabu search. Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[37] GUROBI. Gurobi optimizer. http://www.gurobi.com, 2016.

[38] S. Hartung, C. Komusiewicz, and A. Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. In D. Thilikos and G. Woeg-

inger, editors, *Parameterized and Exact Computation*, volume 7535 of *Lecture Notes in Computer Science*, pages 231–241. Springer, 2012.

[39] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[40] P. Hintsanen. The most reliable subgraph problem. In *Knowledge Discovery in Databases: PKDD 2007*, pages 471–478. Springer, 2007.

[41] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *Data Mining and Knowledge Discovery*, 17(1):3–23, 2008.

[42] R. Jin, L. Liu, and C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 992–1000. ACM, 2011.

[43] S. Kahruman-Anderoglu, A. Buchanan, S. Butenko, and O. A. Prokopyev. On provably best construction heuristics for hard combinatorial optimization problems. *Networks*, 2015. To appear.

[44] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[45] G. Kochenberger, F. Glover, B. Alidaee, and H. Wang. Clustering of microarray data via clique partitioning. *Journal of Combinatorial Optimization*, 10(1):77–92, 2005.

[46] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):325–336, 2013.

[47] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27(2):49–64, 1997.

[48] P. Krokhmal, R. Murphey, P. M. Pardalos, S. Uryasev, and G. Zrazhevski. Robust decision making: Addressing uncertainties in distributions. In S. Butenko, R. Murphey, and P. M. Pardalos, editors, *Cooperative Control: Models, Applications and Algorithms*, volume 1 of *Cooperative Systems*, pages 165–185. Springer, 2003.

[49] E. D. Levy, J. B. Pereira-Leal, C. Chothia, and S. A. Teichmann. 3d complex: a structural classification of protein complexes. *PLoS Comput Biol*, 2(11):e155, 2006.

[50] L. Liu, R. Jin, C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs. In *ICDM*, pages 459–468. Citeseer, 2012.

[51] R. Luce and A. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949.

[52] F. Mahdavi Pajouh and B. Balasundaram. On inclusionwise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization*, 9(2):84–97, 2012.

[53] K. Menger. Zur allgemeinen Kurventheorie, 1927.

[54] Z. Miao, B. Balasundaram, and E. L. Pasiliao. An exact algorithm for the maximum probabilistic clique problem. *Journal of Combinatorial Optimization*, 28:105–120, 2014.

[55] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[56] R. J. Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[57] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[58] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120:197–207, 2002.

[59] J. Palmquist P. Krohmal and S. Uryasev. Portfolio optimization with conditional value-at-risk objective and constraints. *The Journal of Risk*, 2:11–27, 4 2002.

[60] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[61] A. Parley, S. Hedetniemi, and A. Proskurowski. Partitioning trees: matching, domination, and maximum diameter. *International Journal of Computer & Information Sciences*, 10(1):55–61, 1981.

[62] J. Pattillo, N. Youssef, and S. Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

[63] R. T. Rockafellar and S. Uryasev. Optimization of conditional value-at-risk. *The Journal of Risk*, 2:21–42, 3 2000.

[64] R. T. Rockafellar and S. Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking and Finance*, 26(7):1443–1471, 2002.

[65] M. Rysz, M. Mirghorbani, P. Krokhmal, and E. L. Pasiliao. On risk-averse maximum weighted subgraph problems. *Journal of Combinatorial Optimization*, 28(1):167–185, 2014.

[66] S. B. Seidman and B. L. Foster. A graph theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.

[67] S. Shahinpour and S. Butenko. Algorithms for the maximum k-club problem in graphs. *Journal of Combinatorial Optimization*, 26(3):520–554, 2013.

[68] S. Shahinpour and S. Butenko. Dinstance-based clique relaxations in networks: s-cliques and s-clubs. In B. I. Goldengorin, V. A. Kalyagin, and P. M. Pardalos,

editors, *Models, Algorithms, and Technologies for Network Analysis*, volume 59 of *Springer Proceedings in Mathematics & Statistics*, pages 149–174. Spinger Science + Business Media, New York, 2013.

[69] P. Soriano and M. Gendreau. Diversification strategies in tabu search algorithms for the maximum clique problem. *Annals of Operations Research*, 63:189–207, 1996.

[70] E. Tomita and T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In C. Calude, M. Dinneen, and V. Vajnovszki, editors, *Discrete Mathematics and Theoretical Computer Science*, volume 2731 of *Lecture Notes in Computer Science*, pages 278–289. Springer Berlin / Heidelberg, 2003.

[71] A. Veremyev and V. Boginski. Identifying large robust network clusters via new compact formulations of maximum k-club problems. *European Journal of Operational Research*, 218(2):316–326, 2012.

[72] A. Verma, A. Buchanan, and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27:164–177, 2015.

[73] Q. Wu, J. K. Hao, and F. Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196:611–634, 2012.

[74] Q. Wu and J.K. Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 26:86–108, 2013.

[75] Q. Wu and J.K. Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[76] M. Yannakakis. Node- and edge-deletion NP-complete problems. *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, 1978.

[77] H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22:823–829, 2006.

[78] Z. Zou, J. Li, H. Gao, and S. Zhang. Finding top-k maximal cliques in an uncertain graph. In *Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE)*, pages 649–652. IEEE, 2010.

[79] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.

APPENDIX A

PRECOMPUTED VALUES OF CVAR FOR CLIQUE INSTANCES WITH THE SAME EDGE FAILURE PROBABILITY

Table A.1: Values of $CVaR_{0.9}[L_C]$ for various sizes of clique C, where each edge fails with the same probability p.

$\|C\|$	$p = 0.05$	$p = 0.075$	$p = 0.1$	$p = 0.125$	$p = 0.15$	$p = 0.175$	$p = 0.2$
3	1.0725	1.1603	1.28	1.4297	1.6075	1.8116	2
4	1.2715	1.5819	1.9853	2.076	2.1308	2.2054	2.3027
5	1.6367	2.0794	2.1827	2.3452	2.5749	2.877	3.0788
6	2.0699	2.2246	2.5034	2.9237	3.1351	3.2431	3.4016
7	2.157	2.4914	3.0687	3.2128	3.419	3.7325	4.1146
8	2.3013	2.9157	3.2258	3.5125	3.979	4.2101	4.3886
9	2.5286	3.1883	3.5054	4.0517	4.287	4.5417	4.9437
10	2.8272	3.2968	3.8173	4.2598	4.6114	5.1586	5.3284
11	3.122	3.5233	4.1831	4.5144	5.1448	5.3643	5.7524
12	3.2055	3.8579	4.3666	4.9748	5.3634	5.7741	6.2668
13	3.3192	4.167	4.619	5.2694	5.6836	6.2515	6.572
14	3.4602	4.2739	4.9576	5.455	6.1545	6.4999	7.0962
15	3.6549	4.42621	5.2477	5.7927	6.374	6.915	7.4148
16	3.9185	4.6443	5.3955	6.2186	6.6533	7.3213	7.7851
17	4.1587	4.9468	5.6323	6.3855	7.1004	7.5975	8.2975
18	4.2295	5.2172	5.9459	6.6217	7.3719	8.0033	8.5462
19	4.3271	5.3231	6.2584	6.9642	7.6208	8.3661	8.9607
20	4.4429	5.4684	6.3906	7.2956	7.9721	8.6325	9.3914

Table A.2: Value of $CVaR_{0.95}[L_C]$ for various sizes of clique C, where each edge fails with the same probability p.

$\|C\|$	$p = 0.05$	$p = 0.075$	$p = 0.1$	$p = 0.125$	$p = 0.15$	$p = 0.175$	$p = 0.2$
3	1.145	1.3206	1.56	1.8594	2	2	2
4	1.5433	2.0335	2.0789	2.1528	2.2616	2.4108	2.6054
5	2.0482	2.1588	2.3654	2.6904	3.0503	3.0928	3.1575
6	2.1398	2.4492	3.0067	3.1337	3.2702	3.4863	3.8031
7	2.3139	2.9827	3.1826	3.4257	3.8381	4.1206	4.2292
8	2.6027	3.1523	3.4515	4.025	4.2039	4.4202	4.7772
9	3.0571	3.3765	4.0108	4.2377	4.5741	5.0833	5.2145
10	3.1531	3.5936	4.1748	4.5195	5.1353	5.317	5.6568
11	3.244	4.0466	4.3663	5.0288	5.3055	5.7286	6.1931
12	3.4109	4.1947	4.7332	5.2691	5.7268	6.2241	6.5335
13	3.6384	4.334	5.1558	5.5388	6.1943	6.503	7.144
14	3.9204	4.5478	5.265	5.91	6.3901	6.9998	7.4009
15	4.1337	4.8524	5.4954	6.2436	6.7479	7.3232	7.8295
16	4.2136	5.17	5.78	6.4372	7.2247	7.6425	8.2871
17	4.3175	5.2801	6.1942	6.7711	7.4267	8.1951	8.5951
18	4.459	5.4345	6.3257	7.2185	7.7437	8.4022	9.0924
19	4.6543	5.6463	6.5168	7.3692	8.2272	8.7323	9.4083
20	4.8858	5.9368	6.7812	7.5911	8.4026	9.2546	9.7828